UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Sistema de Posicionament en Interiors a través d'una càmera.

Memòria del projecte que presenta Robert Figueroa Moreno

sota la direcció del Dr. Eng. Pere Palà Schönwälder

per assolir el grau d'Enginyer en Sistemes TIC.

Agraïments

En primer lloc m'agradaria agrair l'ajut del meu tutor, el doctor Pere Palà Schönwälder. Ell m'ha ajudat molt realitzant-me un seguiment constant durant tot el projecte, resolent-me dubtes i sobretot aclarir-me els conceptes teòrics. També, a estructurar bé el projecte perquè aquest anés avançant.

També voldria agrair l'ajut del Marcel Masnou, perquè és qui va donar forma al objecte de referència amb un programa de disseny gràfic. Al Jordi Albiol també per imprimir-me les peces en 3D. Agreixo igualment l'ajut i el suport que m'han donat tant els meus companys de carrera com els meus amics.

Per acabar voldria agrair a la meva família donar-me l'ajut i l'esforç que m'han donat per poder acabar el treball i per tant un grau en enginyeria.

Abstract

The purpose of this thesis was to build a system that allow us to know our location in an indoors place. We will use a camera and a referenced object for that. The idea behind is knowing the camera position and orientation depending on how the camera sees the object.

There are two principal goals that we have to achieve:

- Object recognition: We will have to be able to recognize the object inside de picture.
- Calculate the position: Make a relation to get the position knowing the real object's dimensions and the object's dimensions in the photo.

We will use Python as the code language. We will create our own code to make possible the object recognition. And for knowing the position we will write an algorithm, that was presented in one article.

This code will run in a Raspberry Pi 3 with a RaspiCam, which is the camera that fits perfectly with the Raspberry products.

Índex

I. Memòria

1. Introducció

La idea d'aquest projecte va ser donada pel professorat TIC de l'EPSEM. Aquesta, com d'altres, estava publicada dins de les propostes de l'any 2016/2017, però aquesta justament vaig trobar que era molt interessant i un bon tema per desenvolupar durant aquest últim trimestre d'enginyeria TIC.

Aquesta idea, més específicament, ve donada pel meu tutor el doctor Pere Palà Schönwälder. Pel títol, sistema de posicionament en interiors a través d'una càmera, és fàcil desxifrar l'objectiu del treball. Es volia dissenyar un sistema el qual permetés posicionar-te en interiors. Aquest sistema compta amb dos objectes claus, un és una ***figura de dimensions conegudes*** per tenir una referència i l'altre és la ***càmera*** que servirà per localitzar aquesta figura. En funció de com sigui vista, es determinarà la posició relativa de la càmera respecta l'objecte.

En el moment de plantejar el projecte sabíem que principalment hi haurien dos accions a dur a terme. La primera seria el reconeixement de la figura dins la foto (tractament de la imatge) i la següent, un cop coneixem la posició dins de la imatge, determinar la posició relativa respecte la figura.

No és la primera vegada que un TFG tracta el tema de posicionaments en interiors a través d'una càmera [14], per tant tenia una bona base per poder començar a estructurar el treball. Anys anterior s'havia utilitzat llibreries de seguiment d'objectes, però en aquest com a diferència, es volia arribar al mateix objectiu però creant nosaltres les llibreries .

Es va decidir utilitzar el Python com a llenguatge de programació pel desenvolupament del treball ja que durant la carrera s'ha utilitzat bastant i trobo que te eines força potents pel tractament bàsic d'imatge i pels càlculs matemàtics.

Aquest treball té una base de teoria espacial que s'ha d'entendre per poder seguir i comprendre els passos seguits per arribar al nostre objectiu. Per això al principi es començarà explicant-ne conceptes bàsics claus.

2. Plantejament General

En aquest apartat explicarem perquè va sorgir la idea de dissenyar un sistema de posicionament en interiors a través d'una càmera. També s'explicaran les primeres hipòtesis que es van plantejar i perquè al final no van ser possibles dur-les a terme.

2.1. Situació del problema

Molts estareu fent aquesta relació directa entre el posicionament i un sistema GPS. Ara bé la cosa es complica quan aquest posicionament s'ha de fer en interiors ja que la potència del senyal GPS en aquest llocs baixa dràsticament. Per tant s'havia de buscar una alternativa. Com que s'havia pensat que un sistema com aquest seria molt útil per drons i/o robots mòbils i en moltes ocasions aquests en disposaven d'una càmera, vam pensar que utilitzar-la seria molt bona idea.

El sistema que s'havia plantejat hauria de tenir les següents característiques:

- A temps real, en tot moment saber la posició que la càmera es troba respecte l'objecte
- Precís

S'ha agafat com a referència que el sistema hauria d'analitzar com a mínim 3 imatges per segon. Per tant, aproximadament saber a quina distància estem de l'objecte cada 0.33 s. Un altre punt a tenir en compte és la distància màxima, que vindrà determinada pel rang de la càmera, fins on pugui veure aquest objecte. Això vol dir que les distàncies que podrem mesurar no seran gaire grans, inferiors a 5m. Es va decidir que com a màxim la tolerància d'error fos d'un 10%.

En la figura 1, es pot veure la idea que va al darrera d'aquest treball. A més a més ja es comencen a veure conceptes claus com ara els eixos de coordenades. Sempre representarem els eixos de coordenades de la següent manera:

- Eix de les Z en Blau
- Eix de les X en Verd
- Eix de les Y en Vermell

Un altre dels conceptes claus que podem veure en aquesta foto és que l'objecte es trobarà a l'origen de coordenades.

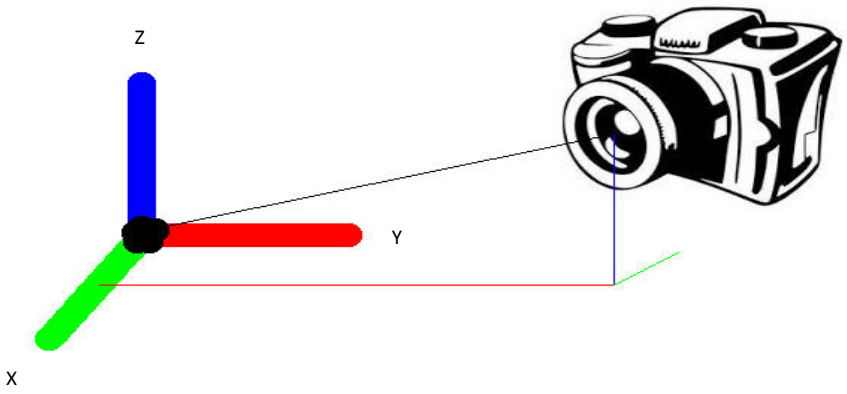


Figura 1: Representació general del sistema de posicionament

2.2. Primeres idees

Primerament vam pensar en fer servir dues càmeres (visió dual) que és el sistema que normalment s'usa per triangular posicions a traves d'imatges i a part d'això també permet la representació en 3D de l'entorn.

Aquesta triangulació s'aconsegueix a través de les transformacions explicades [2]. Com veiem en la figura 2, la "x" representa un punt de l'objecte, les "O" fan referència al centre de la càmera les dues i les dues "y" son les projeccions del punt x en el pla de la imatge de les càmeres . Les dues línies verdes són la unió respectiva entre el centre i el punt projectat al pla de la imatge. I la idea final es que on convergeixin aquestes dues línies serà on es trobi el punt de l'objecte

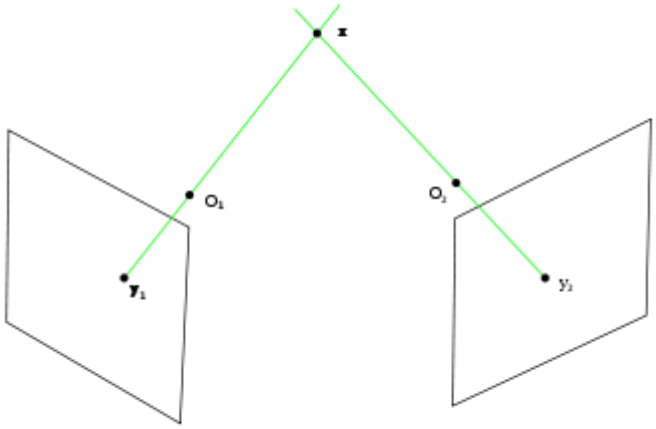


Figura 2: Esquema sobre el procediment de triangulació a traves de dues càmeres

Aquest mètode ens hagués proporcionat la posició i l'orientació, que és l'objectiu, a més d'altre informació que no necessitaríem, com ara obtenir una representació tridimensional de l'entorn. Per contra, el cost de processament i de programació hauria estat més alt. Per tant es va decidir utilitzar només una càmera.

Ara tractarem les diferents formes dels objectes de referència. Primerament es va decidir que el nostre objecte tindria forma de "T" ja que per es una forma poc frqüent de trobar en l'entorn. Per tant en seria molt més fàcil fer-li un seguiment. La primera tenia un forma tal i com podem veure en la figura 3.

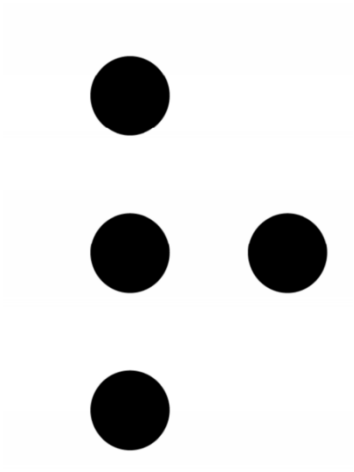


Figura 3: Primer esbós del objecte de referència

Es pot veure que està format per cercles, perquè es va considerar que aniria bé per definir els extrems de la l'objecte en forma de"T". Aquesta figura va ser la que es va fer servir durant molt temps per testejar l'algoritme de reconeixement de l'objecte. El problema va arribar alhora d'implementar l'algoritme per determinar la distancia. Aquest necessitava com a objecte de referència un que fos un quadrilàter on els angles entre els punts fossin inferiors de 180 i més grans que 0, per tant la figura 3 no complia amb aquestes característiques es va haver de dissenyar una altre.

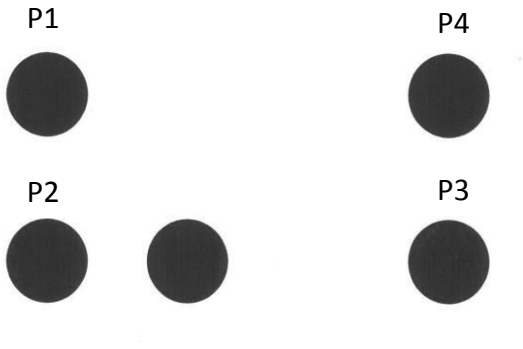


Figura 4: Segon esbós del objecte de referència

Com podem veure en la Figura 4, els punts numerats, formen un quadrilàter amb les condicions abans esmenades. L'ordre dels punts es important i s'explicarà en el apartat (*4.1 Idea general)*, es per això que hi ha un 5è punt que ens servirà per a determinar l'ordre.

Una part important que no he mencionat encara són les mides d'aquesta figura. Tots els cercles tenen un diàmetre de 20 mm. Les següents distàncies faran referència sempre des del centre del cercles. La separació entre els punts número u als quatre és de 100 mm igual que la del tres al dos. Llavors tenim que la distància entre el punt número u i el dos és de 40 mm, igual que la que hi ha de tres i el quatre. Per acaba ja el punt de referència es troba a 35mm del punt numero dos i a 65 mm del punt numero tres.

3. Conceptes teòrics

En aquest apartat s'explicaran els conceptes teòrics bàsics necessaris per a la bona comprensió del treball.

3.1. Eixos de coordenades

Pot ser un dels punts clau del treball, ja que es tracta de trobar la posició (x,y,z) i la orientació en què es trobe la càmera respecte el nostre objecte. Tal i com veiem a la figura 5 el nostre sistema està format per 3 eixos de coordenades diferents, un que li direm el de l'objecte, l'altre el de la càmera i per acabar el del pla de la imatge.

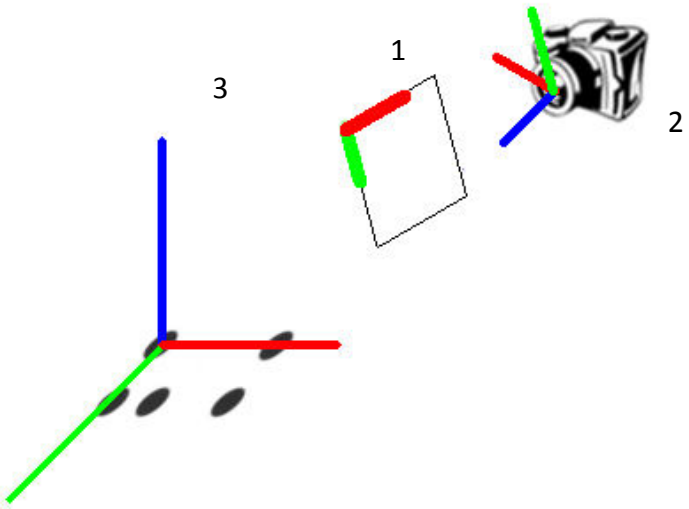


Figura 5: Representació dels diferents eixos de coordenades presents

Per tant, s'ha de tenir clar en tot moment respecte quin dels tres eixos diferents ens estem referenciant. L'odre que seguirem serà el següent:

- (1) Eix del Pla de la càmera: es tindran els punts referenciats a través del pla de la càmera (x,y) amb píxels.
- (2) Eix de la càmera: després efectuarem una primera transformació, en què convertirem els píxels de l'apartat anterior (1) a mil·límetres. Aquest serà el primer canvi de referència.
- (3) Eix de l'objecte: per acabar, s'aplicarà un segona i última transformació amb què finalment obtindrem la posició de l'objecte, objectiu del treball.

3.2. Girs sobre els eixos i transformacions d'eixos

L'orientació estarà definida pels diferents girs que es produeixen sobre els tres eixos del sistema de coordenades. Com a norma general farem servir la regla de la mà dreta pels gir positiu, en la figura 6 veurem representats els girs negatius.

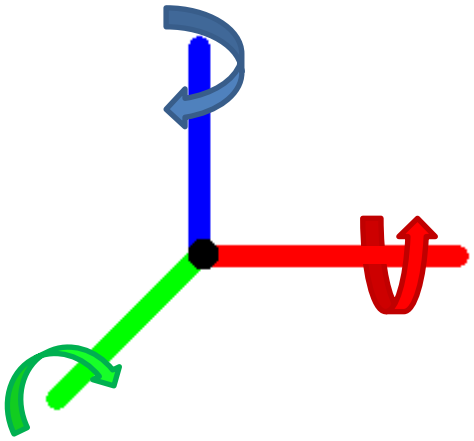


Figura 6 : Girs negatius dels tres eixos per separat

Ara bé, és important saber que sempre els girs els expressem en funció de l'eix original, mai de l'eix resultant. També cal remarcar que les rotacions segueixen un ordre establert. El primer gir serà sobre l'eix de les X, després sobre el de les Y i per acabar, sobre el de les Z. En la Taula 1 podem veure uns exemples de combinació de girs. L'orientació inicial la trobarem representada en la figura 7.

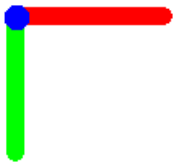
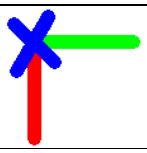
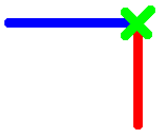


Figura 7: Orientació inicial de l'eix

GIRS APLICAT			ORIENTACIÓ
X	Y	Z	
180º	0º	0º	
180º	0º	90º	
180º	90º	90º	

Taula 1

A la taula 1 acabem de veure el tipus de transformació ***rotació*** . El sistema de coordenades pot rebre un altre tipus de transformació, la ***translació***. Que consisteix en el desplaçament de l'origen de coordenades.

3.3. Projecció d'objectes

En aquest apartat explicarem com un objecte és representat en el pla de la càmera, mitjançant una imatge o fotografia. Aquí també serà important saber algunes característiques de la càmera, com ara la distància focal i també les dimensions del pla imatge.

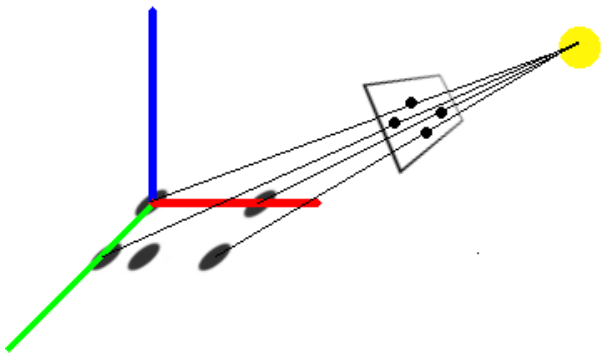


Figura 8: Representació sobre la projecció d'objectes en el pla de la càmera

Com representa la figura 8, s'hi veuen diferents línies que van del centre de la càmera als diferents punts de l'objecte. Veiem que a una certa distància aquestes línies són tallades per un pla. Aquest pla és el de la imatge i la distància, respecte el centre de la càmera, és la distància focal de la càmera. El punt on es talla el pla i la línia li direm la projecció del punt de l'objecte.

De vegades aquest aquet pla no es recte, sinó que presenta una certa curvatura. Això implica que els punts projectats no es troben a la mateixa distància del centre. Pel projecte, aquest efecte seria molt perjudicial per determinar la distància.

3.4. Transformacions dels punts

Un cop hem vist les possibles transformacions que un eix pot patir, anem a veure com afecten aquestes als punts. S'explicarà amb l'ajut d'un exemple pràctic. Per simplificar tindrem el punt P en la posició $(1,1,0)_1$ respecte el sistema de coordenades S_1. El nou origen se situa a la posició $(10,5,20)_1$ amb un girs de $(180^o,0^o,0^o)$.

Aquest punt P tindrà unes coordenades $(-9,4,20)_2$ respecte el nou sistema de coordenades S_2. És un exemple bastant simple ja que els tres plans entre ells són paral·lels. Quan les rotacions no són amb angles múltiples de 90, els càlculs ja són més difícils de ser efectuats visualment.

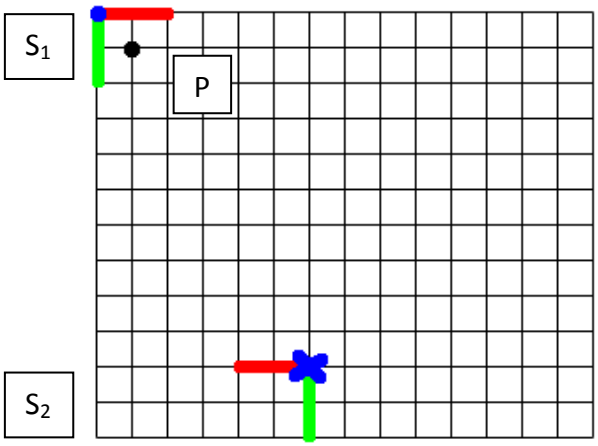


Figura 9:

Per això, el que farem serà trobar una matriu que ens permeti relacionar els punts expressats amb sistemes de coordenades diferents. Aquesta matriu, anomenada T, la trobarem explicada més en profunditat en l'apartat (*4.2 Explicació del procediment*).

4. Algoritme

En aquest aparat s'explicarà el procediment que s'ha seguit per determinar la posició de la càmera un cop el nostre software ha detectat on es troba l'objecte dins de la imatge. Aquest algoritme és el mateix que és emprat per [1] en un problema similar al nostre. El que s'ha fet ha estat fer una tria de les equacions matemàtiques de [1] necessàries per determinar la posició i orientació i reescriure-les amb Python. L'algoritme es troba detallat en el mòdul anomenat ***caluculate_coordinades.py*.**

4.1. Idea general

L'esquema de la figura 10 ens mostra els passos a seguir per determinar la posició i l'orientació de l'objecte.

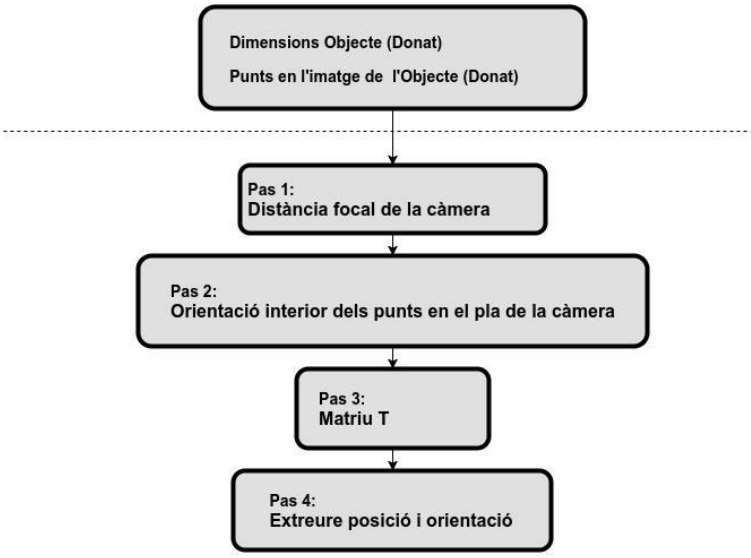


Figura 10: Esquema dels procediments que segueix l'algoritme per determinar la posició

A l'inici de la figura 10, abans de les línies discontinues hi trobem els paràmetres necessaris per a realitzar correctament l'algoritme. Aquests són la posició dels quatre punt en referència del pla de la imatge i les mides originals del quadrilàter. Com són 4 punts, necessitarem 6 distàncies per saber exactament la forma.

El primer pas que indica l'esquema és determinar la distància focal de la càmera. En el nostre cas aquest no el realitzarem ja que fem servir una càmera la qual és fixa i coneguda. El model i les especificacions les trobarem explicades en l'apartat (*6.2 Càmera*).

En el segon, es determinen les coordenades dels punts de l'objecte respecte els eixos de coordenades de la càmera. A part d'això, l'algoritme ens permet també saber les distàncies absolutes de cada punt respecte la càmera. Per les comprovacions pràctiques del sistema serà molt més fàcil determinar si l'algorisme funciona bé amb aquestes distàncies que no pas amb els punts representats respecte els eixos de coordenades de la càmera.

En el tercer pas, es troba la matriu que hem anomenat anteriorment **T**, o també anomenada **Transformation Matrix [**3]. Aquesta és una matriu que relaciona punts entre dos sistemes de coordenades diferents. Per tant, si tenim un punt P_1 referenciat amb un sistema de coordenades, S_1, multiplicant aquest punt per la matriu ens donarà el mateix punt, P_2, però ara referenciat amb un altre sistema de coordenades, S_2, segons $P_2 = T * P_1$. La figura 11 representa la transformació entre els dos sistemes S_1 i S_2.

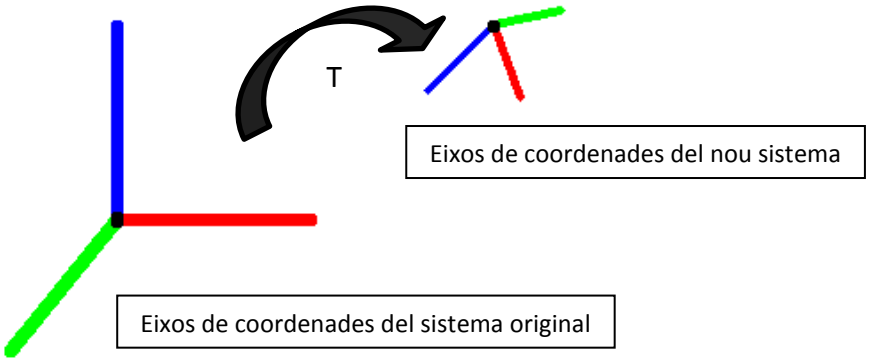


Figura 11: Representació de la matriu T

La matriu **T** està formada per les diferents rotacions dels tres eixos, $\boldsymbol{R_\alpha}$ sobre l'eix X representada en equació (*2.2*), $\boldsymbol{R_\beta}$ sobre l'eix Y representada en equació(*2.3*), $\boldsymbol{R_\gamma}$ sobre l'eix Z representada en l'equació (*2.4*) i la translació D representada en l'equació (*2.5*). L'ordre dels les rotacions és fonamental.

$$T = R_\alpha R_\beta R_\gamma D \qquad (2.1)$$

$$R_\alpha = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos_\alpha & -sin_\alpha & 0 \\ 0 & sin_\alpha & cos_\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.2)$$

$$R_\beta = \begin{bmatrix} cos_\beta & 0 & -sin_\beta & 0 \\ 0 & 1 & 0 & 0 \\ sin_\beta & 0 & cos_\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

$$R_\gamma = \begin{bmatrix} cos_\gamma & sin_\gamma & 0 & 0 \\ -sin_\gamma & cos_\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

$$D = \begin{bmatrix} 1 & 0 & 0 & X_0 \\ 0 & 1 & 0 & Y_0 \\ 0 & 0 & 1 & Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

Els punts $\boldsymbol{X_i^C\ Y_i^C\ Z_i^C, i = 1,2,3,4}$ són els quatre punts del nostre objecte en relació els sistema de coordenades de la càmera i els $\boldsymbol{X_i^O\ Y_i^O\ Z_i^O, i = 1,2,3,4}$ són els mateixos quatre punts, però en relació els eixos de coordenades de l'objecte. Els $\boldsymbol{X_i^O\ Y_i^O\ Z_i^O}$ ja són coneguts abans d'aplicar l'algoritme, i els $\boldsymbol{X_i^C\ Y_i^C Z_i^C}$ s'han de calcular.

$$W_i^C = T\, W_i^O, i = 1,2,3,4 \quad (2.6)$$

$$\begin{bmatrix} X_1^C & X_2^C & X_3^C & X_4^C \\ Y_1^C & Y_2^C & Y_3^C & Y_4^C \\ Z_1^C & Z_2^C & Z_3^C & Z_4^C \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1^O & X_2^O & X_3^O & X_4^O \\ Y_1^O & Y_2^O & Y_3^O & Y_4^O \\ Z_1^O & Z_2^O & Z_3^O & Z_4^O \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (2.7)$$

L'equació(*2.6*) mostra la relació entre els punts expressats en sistemes de coordenades diferents. On $\boldsymbol{W}$ és un conjunt de punts

Per acabar, un cop trobem aquesta matriu **T**, el que farem serà extraureure els valors de $\boldsymbol{X_0, Y_0, Z_0, \alpha, \beta, \gamma}$.

4.2. Explicació del procediment

A la taula 2 podem veure definits el diferents símbols que utilitzarem durant l'explicació de l'algoritme.

Símbol	Definició
$\boldsymbol{A_i}$	Àrea del triangle format per $\Delta(P_i, P_j, P_k)$.
$\boldsymbol{B_i}$	Dos vegades l'àrea formada per tres dels quatre punts de la imatge.
$\boldsymbol{C}$	Centre de la projecció(càmera)
$\boldsymbol{C_{ij}}$	Calcula la relació entre $\boldsymbol{C}$ i l'objecte.
$\boldsymbol{d_i}$	Distància absoluta que hi ha entre $\boldsymbol{C}$ i P_i.
$\boldsymbol{f}$	Distància focal.
$\boldsymbol{F_i}$	Mòdul del vector $\boldsymbol{Q_i^f}$.
$\boldsymbol{H_{ij}}$	Calcula distància focal efectiva.
$\boldsymbol{h}$	Altura des de $\boldsymbol{C}$ **fins al centre de l'objecte**
$\boldsymbol{P_i}$	Punts de l'objecte.
$\boldsymbol{P_i^f}$	Vector que va de $\boldsymbol{C}$ a $\boldsymbol{P_i}$.
$\boldsymbol{Q_i^C}$	Punts dels objectes projectats en el pla de la imatge.
$\boldsymbol{Q_i^f}$	Vector que va de $\boldsymbol{C}$ a $\boldsymbol{Q_i^C}$.
$\boldsymbol{R_{ij}}$	Calcula posició relativa dels diferents $\boldsymbol{Q_i^C}$.
$\boldsymbol{s_{ij}}$	Longitud del segment $P_i\ P_j$.
$\boldsymbol{T}$	Transformation Matrix
$\boldsymbol{u_i}$	Vector unitària del vector $\boldsymbol{P_i^f}$
$\boldsymbol{V_i}$	Volum del tetraedre format per $\boldsymbol{\tau(C, P_i, P_j, P_k)}$
$\boldsymbol{X_i^f\ Y_i^f\ Z_i^f}$	Components x,y,z del P_i^f
$\boldsymbol{\tau(C, P_i, P_j, P_k)}$	Tetraedre format pel punts $\boldsymbol{C}, P_i, P_j, P_k$.
$\Delta(\boldsymbol{P_i, P_j, P_k})$	Àrea formada pels punts P_i, P_j, P_k

Taula 2

A la figura 12 es pot veure un esquema general del problema. A la banda dreta de la figura trobem els diferents punts $\boldsymbol{P_i}$**.** En el centre trobem el punt $\boldsymbol{C}$ amb l'orientació del eixos de coordenades de la càmera ($\boldsymbol{X_c^f}$, $\boldsymbol{Y_c^f}$, $\boldsymbol{Z_c^f}$). I per acabar, en la zona esquerra de la foto hi observem els diferents $\boldsymbol{Q_i^C}$.

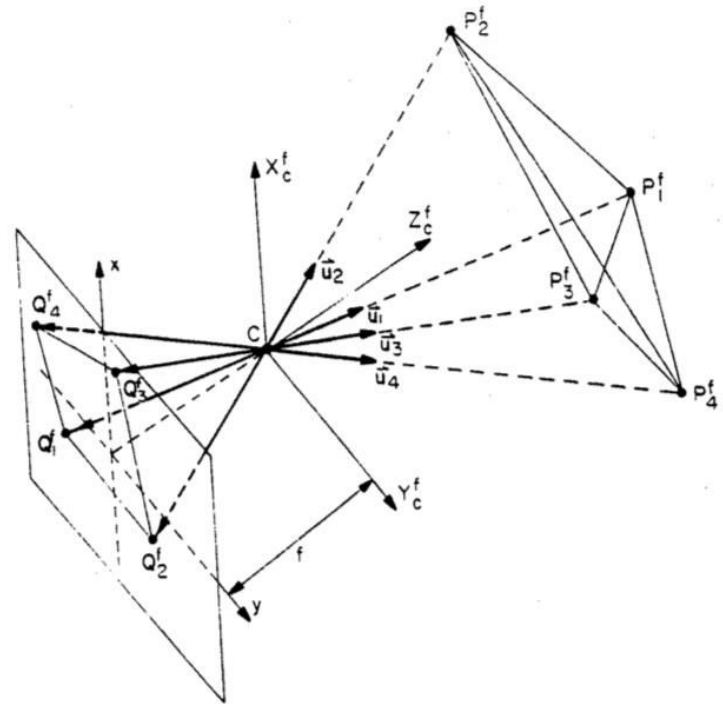


Figura 12: Imatge general dels elements de l'algoritme

Primer de tot es calcula les diferents quatre ares possibles que forma els triangles del nostre objecte, la part dreta de la fotografia, i després els volums que formen amb el punt $\boldsymbol{C}$.

$$A_1 = \Delta(P_1, P_2, P_3) = [(s_{12}^2 + s_{13}^2 + s_{23}^2)^2 - 2(s_{12}^2 + s_{13}^2 + s_{23}^2)^2]^{1/2}/4 \quad (2.8)$$

$$A_2 = \Delta(P_1, P_2, P_4) = [(s_{12}^2 + s_{14}^2 + s_{24}^2)^2 - 2(s_{12}^2 + s_{14}^2 + s_{24}^2)^2]^{1/2}/4 \quad (2.9)$$

$$A_3 = \Delta(P_1, P_3, P_4) = [(s_{13}^2 + s_{14}^2 + s_{34}^2)^2 - 2(s_{13}^2 + s_{14}^2 + s_{34}^2)^2]^{1/2}/4 \quad (2.10)$$

$$A_4 = \Delta(P_2, P_3, P_4) = [(s_{23}^2 + s_{24}^2 + s_{34}^2)^2 - 2(s_{23}^2 + s_{24}^2 + s_{34}^2)^2]^{1/2}/4 \quad (2.11)$$

$$V_1 = \tau(C, P_1, P_2, P_3) = hA_1/3 \quad (2.12)$$

$$V_2 = \tau(C, P_1, P_2, P_4) = hA_2/3 \quad (2.13)$$

$$V_3 = \tau(C, P_1, P_3, P_4) = hA_3/3 \quad (2.14)$$

$$V_4 = \tau(C, P_2, P_3, P_4) = hA_4/3 \quad (2.15)$$

Fins ara totes les variables que s'han fet servir són conegudes, menys la $\boldsymbol{h}$. Per poder progressar amb l'algoritme començarem a tractar els punt en el pla de la imatge.

Tal com es veu en la equació (*2.16*) i en la figura 12, el pla de la imatge esta separats paral·lelament a una distància igual a la f de l'eix de les Z . Els punts (x_i, y_i) són les coordenades del punt en el pla de la imatge.

$$Q_i^C = (x_i, y_i, -f) \qquad (2.16)$$

Com s'explica en l'apartat (*3.3. Projecció d'objectes*) el punt projectat ($\boldsymbol{Q_i^C}$) està situat sobre una línia on formada pel centre de la càmera ($\boldsymbol{C}$) i el punt en la realitat ($\boldsymbol{P_i}$**).** S'observen dins la figura 12 que els diferents vectors $\boldsymbol{Q_i^f}$ tenen la mateixa direcció que els u_i però en sentit contrari.

$$u_i = (-x_i, -y_i, f)/F_i \qquad (2.17)$$

$$F_i = \sqrt{x_i^2 + y_i^2 + f^2} \qquad (2.18)$$

Per tant amb aquestes dues equacions (*2.17, 2.18*) deduïm la següent:

$$P_i^f = d_i u_i \qquad (2.19)$$

Ara es combinaran les equacions (*2.12, 2.13, 2.14, 2.15*) amb la nova manera que s'ha trobat per expressar els $\boldsymbol{P_i^f}$ en l'equació (2.19). Per això es necessari abans expressar una nova forma de calcular el volum d'un tetraedre, aquesta vegada en funció dels vectors que van des d'un punt fins els altres 3.

$$V_1 = \tau(C, P_1, P_2, P_3) = \frac{hA_1}{3} = \frac{\left|P_1^f \cdot \left(P_2^f \times P_3^f\right)\right|}{6} = d_1 d_2 d_3 |u_1 \cdot (u_2 \times u_3)|/6 \qquad (2.20)$$

Aplicant aquest canvi a les quatre formules dels volums, equacions(*2.12...2.15*) diferents es troba l'altura a traves de 4 equacions diferents.

$$h = d_1 d_2 d_3 \frac{f}{2F_1F_2F_3}\frac{B_1}{A_1} \qquad (2.21)$$

$$h = d_1 d_2 d_4 \frac{f}{2F_1F_2F_4}\frac{B_2}{A_2} \qquad (2.22)$$

$$h = d_1 d_3 d_4 \frac{f}{2F_1F_3F_4}\frac{B_3}{A_3} \qquad (2.23)$$

$$h = d_2 d_3 d_4 \frac{f}{2F_2F_3F_4}\frac{B_4}{A_4} \qquad (2.24)$$

On les diferents $\boldsymbol{B_i}$:

$$B_1 = x_1(y_3 - y_2) + y_1(x_2 - x_3) + y_2x_3 - x_2y_3) \quad (2.25)$$

$$B_1 = x_1(y_4 - y_2) + y_1(x_2 - x_4) + y_2x_4 - x_2y_4) \quad (2.26)$$

$$B_1 = x_1(y_4 - y_3) + y_1(x_3 - x_4) + y_3x_4 - x_3y_4) \quad (2.27)$$

$$B_1 = x_2(y_4 - y_3) + y_1(x_3 - x_4) + y_3x_4 - x_3y_4) \quad (2.28)$$

Les variables importants són les diferents $\boldsymbol{d_i}$ i no pas l' $\boldsymbol{h}$**.** Per aquest motiu, s'igualaran les equacions (*2.21, 2.22, 2.23, 2.24*) per trobar les relacions que hi ha entre les distàncies. Aquest n'és el resultat:

$$d_2 = \frac{B_3A_4F_2}{A_3B_4F_1}d_1 = C_{12}\frac{F_2}{F_1}d_1 \quad (2.29)$$

$$d_3 = \frac{B_2A_4F_3}{A_2B_4F_1}d_1 = C_{13}\frac{F_3}{F_1}d_1 \quad (2.30)$$

$$d_4 = \frac{B_1A_4F_4}{A_1B_4F_1}d_1 = C_{14}\frac{F_4}{F_1}d_1 \quad (2.31)$$

$$d_3 = \frac{B_2A_3F_3}{A_2B_3F_2}d_2 = C_{23}\frac{F_3}{F_2}d_2 \quad (2.32)$$

$$d_4 = \frac{B_1A_3F_4}{A_1B_3F_2}d_2 = C_{24}\frac{F_4}{F_2}d_2 \quad (2.33)$$

$$d_4 = \frac{B_1A_2F_4}{A_1B_2F_3}d_3 = C_{34}\frac{F_4}{F_3}d_3 \quad (2.34)$$

Es pot notar una certa redundància alhora de calcular les distancies, per tant, per tenir una solució més acurada es calcularan totes i es farà la mitja, respectivament. Ara be, ens faltaria encara trobar el valor de $\boldsymbol{d_1}$ i el de $\boldsymbol{f}$, per poder calcular les diferents $\boldsymbol{F_i}$. En el nostre cas, la $\boldsymbol{f}$ ja és coneguda, però sino ho fos s'hauria de seguir el següents passos.

En l'article[1] s'explica que hi ha 12 formes possibles per poder determinar la distància focal, degut a que per calcular-la necessitem 2 segments del tipus $\left(\boldsymbol{P_iP_j}, \boldsymbol{P_iP_k}\right)$ $\boldsymbol{i \neq j \neq k}$. Nosaltres explicarem el procediment pels segments ($\boldsymbol{P_1P_2}, \boldsymbol{P_1P_3}$).

Ara el que farem serà escriure el quadrat de les dues distàncies d'aquest segments en funció dels dos vectors $\boldsymbol{P_1^f}$ $\boldsymbol{P_2^f}$ i el resultat el combinarem amb equació (*2.29*) per tenir ja la $\boldsymbol{d_1}$.

$$s_{ij}^2 = \left(X_j^f - X_i^f\right) + \left(Y_j^f - Y_i^f\right) + \left(Z_j^f - Z_i^f\right) \quad (2.35)$$

$$s_{ik}^2 = \left(X_k^f - X_i^f\right) + \left(Y_k^f - Y_i^f\right) + \left(Z_k^f - Z_i^f\right) \quad (2.36)$$

On:

$$X_i^f = u_{ix} * t = \frac{-x_i}{F_i} * t,\ Y_i^f = u_{iy} * t = \frac{-y_i}{F_i} * t,\ Z_i^f = u_{iz} * t = \frac{f}{F_i} * t \quad (2.37)$$

La $\boldsymbol{t}$ com podem deduir tindrà el valor de $\boldsymbol{d_i}$ ja que serà la distància on es troba $\boldsymbol{P_i^f}$. Si substituïm el valors definit en la equació(*2.37*) en les dues equacions(*2.35, 2.36*) i després substituïm la $\boldsymbol{d_i}$ per la corresponent equacions (*2.29 ... 2.34*) ens quedaria un resultat semblant al següent:

$$s_{ij}^2 = d_i^2 \left[\left(x_i - C_{ij}x_j\right)^2 + \left(y_i - C_{ij}y_j\right)^2 + f^2\left(1 - C_{ij}\right)^2 \right] / F_i^2 \quad (2.38)$$

$$s_{ik}^2 = d_i^2 [(x_i - C_{ik}x_k)^2 + (y_i - C_{ik}y_k)^2 + f^2(1 - C_{ik})^2] / F_i^2 \quad (2.39)$$

Si definim:

$$H_{ij}^2 = \left(x_i - C_{ij}x_j\right)^2 + \left(y_i - C_{ij}y_j\right)^2 \quad (2.40)$$

Per trobar la distància focal haurem d'aïllar $\boldsymbol{f}$ les equacions(*2.38, 2.39*). I aquest n'és el resultat.

$$f = \sqrt{\left[\frac{s_{13}^2 H_{12}^2 - s_{12}^2 H_{13}^2}{s_{12}^2(1-C_{13})^2 - s_{13}^2(1-C_{12})^2}\right]} \quad (2.41)$$

En l'article [1] es mencionen onze expressions més poder trobar la $\boldsymbol{f}$,a part de la que surt a l'equació (*2.42*). Aquesta redundància s'utilitzarà, també, per trobar un resultat més acurat. Utilitzant aquestes dotze formes d'expressar la $\boldsymbol{f}$ i les equacions(*2.38 i 2.39*) ens surt que la $\boldsymbol{d_1}$ es pot expressar de sis formes equivalents:

$$\left\{ \begin{array}{r} d_1 = s_{12}F_1R_{12}^{-1} \\ s_{13}F_1R_{13}^{-1} \\ s_{14}F_1R_{14}^{-1} \\ s_{23}F_1R_{23}^{-1}/C_{12} \\ s_{24}F_1R_{24}^{-1}/C_{12} \\ s_{34}F_1R_{34}^{-1}/C_{13} \end{array} \right\} \quad (2.42)$$

Un cop ja tenim el valor de d_1 ja podem calcules les tres restants amb les equacions (2.29..2.34)

Per poder trobar la matriu T, observant la equació (*2.7*), es veu la necessitat dels 4 punts expressat en funció de l'eix de coordenades de la Coneixent que :

$$P_i^c = d_i u_i = P_i^f + (0,0,f)$$

Deduïm:

$$P_1^c = s_{12}R_{12}^{-1}(-x_1, -y_1, f)' + (0,0,f)' \quad (2.43)$$

$$P_2^c = C_{12}s_{12}R_{12}^{-1}(-x_2, -y_2, f)' + (0,0,f)' \quad (2.44)$$

$$P_3^c = C_{13}s_{12}R_{12}^{-1}(-x_3, -y_3, f)' + (0,0,f)' \quad (2.45)$$

$$P_4^c = C_{14}s_{12}R_{12}^{-1}(-x_4, -y_4, f)' + (0,0,f)' \quad (2.46)$$

On:

$$R_{ij} = H_{ij}^2 + f^2\left(1 - C_{ij}\right)^2 \quad (2.47)$$

Per poder trobar la translació i l'orientació calen que els punts de l'objecte respecte el sistema de coordenades de l'objecte compleixin els següents requeriments:

- Tots els punts han d'estar al pla $\boldsymbol{X^O Y^O}$
- El $\boldsymbol{P_1}$ ha d'estar situat a l'origen de coordenades
- El $\boldsymbol{P_2}$ ha d'estar situat sobre l'eix de les $\boldsymbol{X^O}$
- El $\boldsymbol{P_3}$ ha d'estar situat al primer o al segon quadrant del pla $\boldsymbol{X^O Y^O}$
- El $\boldsymbol{P_4}$ ha d'estar situat a qualsevol punt del pla $\boldsymbol{X^O Y^O}$

Gràcies als requeriments anteriors els valors $\boldsymbol{X_1^O, Y_1^O, Z_1^O, Y_2^O, Z_2^O, Z_3^O, Z_4^O}$ seran 0. Els elements de dins de la matriu **T** els trobarem de la següent manera:

$$W^C = TW \quad (2.48)$$

$$(W^C)' = (W)'T' \quad (2.49)$$

$$\begin{bmatrix} X_1^C & Y_1^C & Z_1^C & 1 \\ X_2^C & Y_2^C & Z_2^C & 1 \\ X_3^C & Y_3^C & Z_3^C & 1 \\ X_4^C & Y_4^C & Z_4^C & 1 \end{bmatrix} = \begin{bmatrix} T_{11} & T_{21} & T_{31} & 0 \\ T_{12} & T_{22} & T_{32} & 0 \\ T_{13} & T_{23} & T_{33} & 0 \\ T_{14} & T_{24} & T_{34} & 1 \end{bmatrix} \begin{bmatrix} X_1^O & Y_1^O & Z_1^O & 1 \\ X_2^O & Y_2^O & Z_2^O & 1 \\ X_3^O & Y_3^O & Z_3^O & 1 \\ X_4^O & Y_4^O & Z_4^O & 1 \end{bmatrix} \quad (2.50)$$

$$T_{11} = \frac{Y_4^O(X_3^C - X_1^C) - Y_3^O(X_4^C - X_1^C)}{X_3^O Y_4^O - X_4^O Y_3^O} \quad (2.51)$$

$$T_{21} = \frac{Y_4^O(Y_3^C - Y_1^C) - Y_3^O(Y_4^C - Y_1^C)}{X_3^O Y_4^O - X_4^O Y_3^O} \quad (2.52)$$

$$T_{31} = \frac{Y_4^O(Z_3^C - Z_1^C) - Y_3^O(Z_4^C - Z_1^C)}{X_3^O Y_4^O - X_4^O Y_3^O} \quad (2.53)$$

$$T_{12} = \frac{X_3^O(X_4^C - X_1^C) - X_4^O(X_3^C - X_1^C)}{X_3^O Y_4^O - X_4^O Y_3^O} \quad (2.54)$$

$$T_{22} = \frac{X_3^O(Y_4^C - Y_1^C) - X_4^O(Y_3^C - Y_1^C)}{X_3^O Y_4^O - X_4^O Y_3^O} \quad (2.55)$$

$$T_{32} = \frac{X_3^O(Z_4^C - Z_1^C) - X_4^O(Z_3^C - Z_1^C)}{X_3^O Y_4^O - X_4^O Y_3^O} \quad (2.56)$$

$$T_{14} = X_1^C \quad (2.57)$$

$$T_{24} = Y_1^C \quad (2.58)$$

$$T_{24} = Z_1^C \quad (2.59)$$

Com que la matriu 3x3 situada a dalt i a l'esquerra de la **T** és ortogonal :

$$T_{13} = T_{21}T_{32} - T_{22}T_{31} \quad (2.60)$$

$$T_{23} = T_{12}T_{31} - T_{11}T_{32} \quad (2.61)$$

$$T_{33} = T_{11}T_{22} - T_{12}T_{21} \quad (2.62)$$

Un cop trobats el valors de la matriu **T**, faltarà obtenir els valors $\boldsymbol{X_0}, \boldsymbol{Y_0}, \boldsymbol{Z_0}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$. En aquest articles es proposava una mètode per trobar-los, però els resultats no acabaven de ser del tot correctes. L'obtenció d'aquest 6 valors es veurà explicada en l'apartat (*5.3. Mòdul per obtenir les distàncies*)

5. Software

En aquest apartat s'explicarà en detall el codi implementat. Abans de tot m'agradaria recordar dir que el llenguatge utilitzat ha estat el Python, ja que és un dels que s'ensenya amb bastanta profunditat en la carrera d'enginyeria en Sistemes TIC. També s'ha triat, perquè te bastantes llibreries que fa molt amigable treballar tant amb el tractament d'imatge com la resolució d'operacions matemàtiques.

Pel tractament d'imatge, Python, disposa d'una llibreria molt potent anomenada OpenCV que disposa de funcions pel reconeixement d'objectes, fer-los un seguiment i moltes altres funcionalitats. Però, junt amb el meu tutor, es va decidir no utilitzar-la degut a que volíem controlar pas a pas com es tractaven les imatges per llavors poder-les optimitzar per millorar el temps de processat.

Vam utilitzar una llibreria que s'anomena Pillow, la qual ens permet llegir una imatge i transformar-la en una matriu de colors. Pel resoldre equacions, matrius i/o expressions matemàtiques utilitzarem una anomenada Numpy. A part d'aquestes dues, es necessitarà una altre que ens permetrà interactuar amb la càmera, i aquesta s'anomena Picamera. Ja per acabar també s'utilitzarà una anomenada Shapely que ens ajudarà molt en un punt del tractament de l'objecte.

Els passos que s'han de seguir per poder instal·lar les diverses llibreries en el nostre projecte les trobarem a la següents referències:

- Pillow [4]
- NumPy [5]
- Picamera [6]
- Shapely [8]

Tot els mòduls implementats els podreu trobar a [13]

5.1. Introducció als mòduls

En els següent apartat(*5.2. Mòdul per reconeixement de l'objecte*) com be diu el seu títol s'explicarà com s'ha fet per poder reconèixer el nostre objecte. Com a concepte general s'ha de tenir clar que aquest mòdul nomes funciona si en l'entorn de l'objecte no hi ha cap objecte negre tret del que fam servir de referencia. El nostre algoritme el que farà serà recorre la imatge en busca de taques negres, que seran els cercles que formen en nostre objecte. Pararà quan n'hagi trobat 5, perquè com veiem a la figura 4 es el nombre de cercles que hi ha. Per tant si hi ha algun altre objecte de color negre el reconeixement no serà correcte. La informació que volem obtenir del cercles és el seu centre i els seus extrems.

En l'apartat (*5.3. Mòdul per obtenir les distàncies*) el que farem serà escriure les formules, equacions de l'apartat (*4. Algoritme*) amb llenguatge Python.

5.2. Mòdul per reconeixement de l'objecte

El mòdul definitiu que farem servir s'anomena *frameV2.py*, ja que es va fer una millora respecte la versió anterior. Aquesta anterior, ens permetia detectar on es trobaven els cercles negres i els seus centres corresponents. Ara be, si el que veiem representat a la foto no eren cercles sino el·lipses o be altres formes, aquesta no funcionava del tot be.

Abans de començar a explicar com tractem la imatge, m'agradaria comentar el procediment sobre com obtenim informació a traves d'ella. Com s'ha mencionat prèviament, utilitzarem de la llibreria Pillow. Aquesta ens permetrà transformar una imatge en una matriu MN on M són els píxels que te la imatge d'altura i N els que té d'amplada.

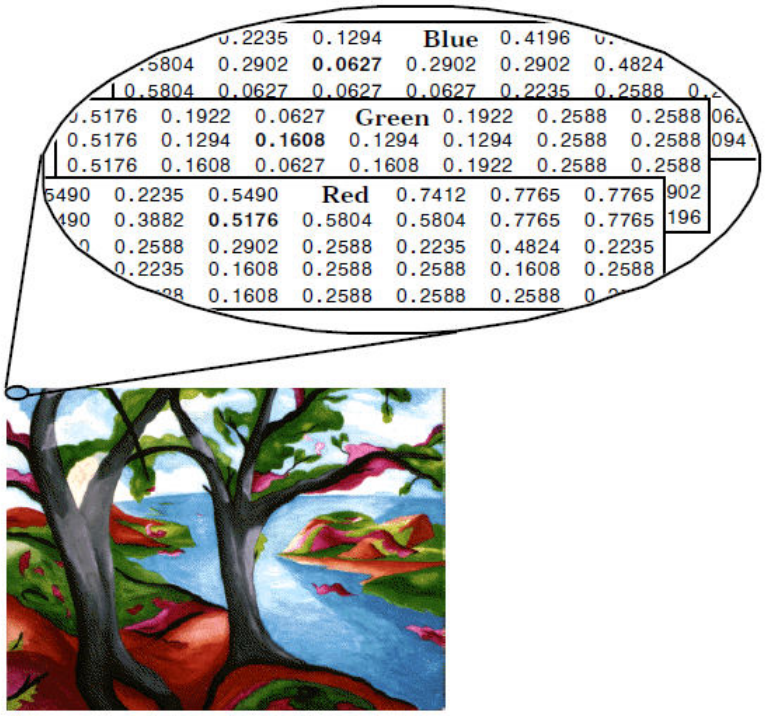


Figura 13: Representació dels valors dels píxels d'una imatge

Com podem veure a la figura 13 cada píxel tindrà una tupla de tres valors (R,G,B). Els quals tindran valors entre 0 i 255, on 0 vol dir l'absència d'aquest color i 255 presencia total. R representa el vermell("red" en anglès) G representa el verd("green" en anglès) i B el blau("blue" en anglès).

El mòdul està format principalment per una classe anomenada ***Frame***, anomenada així ja que el que farem en aquesta classe serà analitzar els frames que em vinguin donats per la càmera.

Aquesta classe disposa de 4 variables i varies funcions. A continuació veurem que volen dir cada variable i perquè s'utilitza:

```
def __init__(self):
        self.PixelMap = []
        self.Centers = []
        self.CirclesInfo = []
        self.OriginalMap = []
```

- **PixelMap**: En aquesta variable s'hi troba guardada la matriu de píxels corresponent a la imatge filtrada ("threshold"). Aquesta l'utilitzarem per fer la recerca dels cercles.
- **Centers**: En aquesta variable s'hi troba guardada un llista de píxels que representen la posició on es troben els diferents centres de les circumferències trobades.
- **CirclesInfo**: En aquesta variables hi haurà una llista de llistes de píxels, on hi hauran els píxels del punt més alt, el punt més baix, el punt més a la esquerra i el punt més a la dreta de les circumferències trobades. El que vindrien a ser els extrems.
- **OriginalMap**: En aquesta variable s'hi troba guardada la matriu de píxels corresponent a la imatge sense filtre. Aquesta s'utilitza bàsicament per debugar visualment el tractament de la imatge.

IMPORTANT: El **Center[i]** es correspon amb la informació del **CirclesInfo[i]**

Aquest mòdul tractarà d'omplir correctament les variables **Center** i **CirclesInfo**. Ara veurem com obtenim les variables **PixelMap** i **OriginalMap**:

En la següent funció es fa servir el mòdul Image de Pillow, la documentació d'aquest la trobareu a la referència [7]. Es guarda la matriu de la imatge en dos variables diferents, una d'elles es filtrarà en funció del valor **thresh**. Això ho fem perquè pot ser que els punts del nostre objecte no siguin un negre (0,0,0) absolut. Per tant es considera negre tots els punts que tinguin valors inferior a aquest **thresh**.

```
def ReadFrame(self, FileName, thresh):
        image = Image.open(FileName)
        OriginalPix = image.load()
        OX, OY = image.size[0], image.size[1]
        image = image.point(lambda i: 255 if i > thresh else 0)
        pix = image.load()
        X, Y = image.size[0], image.size[1]
        data = [[pix[x,y] for x in range(X)] for y in range(Y)]
        OriginalData = [[OriginalPix[x,y] for x in range(OX)] for y in range(OY)]
        self.PixelMap = data
        self.OriginalMap = OriginalData
```

Per debugar farem servir aquest tres funcions següents:

- **Save():** Funció que el que farà serà guardar en un nou fitxer jpg la informació que hi ha en la matriu **OriginalMap.**
- **Save_th():**Funció que el que farà serà guardar en un nou fitxer jpg la informació que hi ha en la matriu **PixelMap.**
- **DrawInImage(ArrayPixels,thickness):** Funció que en permetrà canviar el els píxels de l'**ArrayPixels** a color verd. Es va veure que si nomes pintava un píxel a simple vista no es podia apreciar, però amb una línia en canvi, sí. Per tant es va afegir un nou argument **thickness** que ens permetrà triar la longitud de la línia vertical que pintarà.

Un exemple del resultat obtingut utilitzant les funcions anteriors el podríem veure en la figura 14.

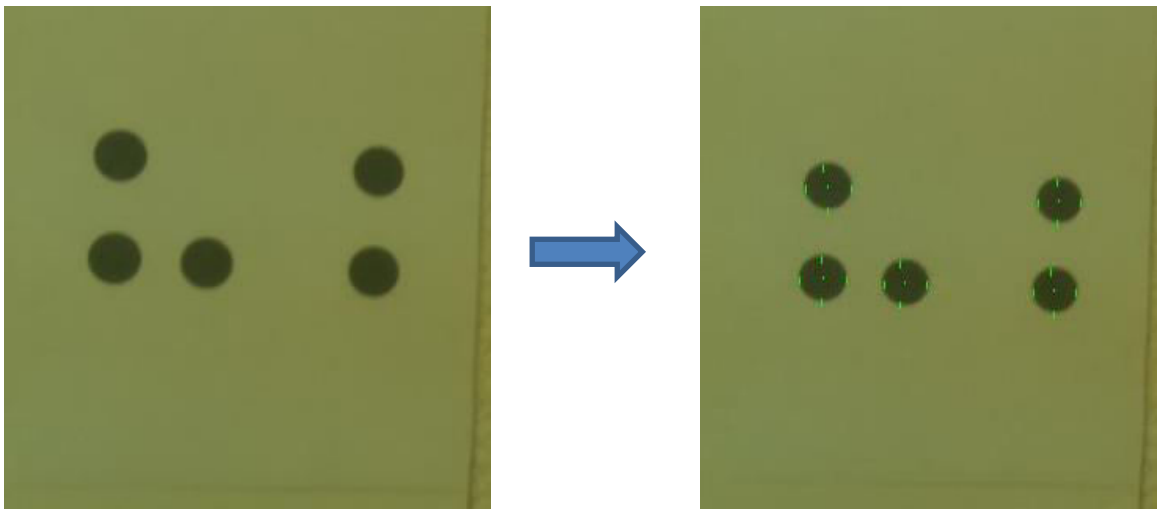


Figura 14: Exemple sobre com visualitzem els resultats

Abans d'entrar a explicar el codi, es farà una breu comentari sobre el plantejament.

Repeteixo, un requeriment essencial que requereix aquest algoritme és que les úniques parts negres que hi poden haver a la imatge siguin les de les circumferències. Cal considerar que no sempre veurem els cercles de forma cicular, ja que al inclinar el pla de la càmera els deforma. Per tant s'havia de pensar un algoritme que funciones per tot tipus de punts negres.

Figura 15: Procediment per detectar centres i extrems de les taques negres

Es pot veure un a seqüència dels passos que seguirem per a la obtenció dels centres i dels extrems en la figura 15. Primer es farà en escombrat d'esquerra a dreta i de dalt a baix de la imatge fins trobar un píxel negre. Un cop trobat, començarà una crida recursiva que consisteix en trobar els extrems. Que vol dir això, doncs buscar el píxel més a la dreta, més a l'esquerra, més amunt i més avall, negre. Els quatre punts que delimiten el punt inicial dins la taca. Aquesta crida es para un cop el punt es troba simultàniament al mig de la línia formada per l'extrem dret i l'extrem esquerra i la que formen els extrem superior i l'inferior. A cada crida els quatre valors dels extrems s'aniran actualitzant, si els algun d'ells sobrepassen els valors anteriors guardats.

Aquest quatre valors es guarden per poder delimitar la zona del cercles, i així si es troba un píxel negre que es troba dins d'un dels cercles ja detectats l'ignorarem. Gràcies a aquest canvi, els temps de detecció va baixar significativament.

Es van trobar casos on amb el primer mostreig no s'acaba de delimitar el total del cercle, un exemple seria el que veiem en la figura 16. Això passava quan la forma era dels cercles es veia força ovalada.

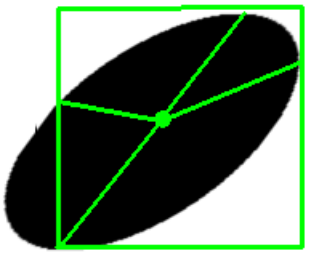


Figura 16: Detectar centre de l'oval

Per tant un cop detectava els píxels negres, fora del rang, aquests el tractava com si fos un cercle nou i quan veiem el resultat del centres n'hi havien de repetits. Per tant vam decidir de tractar aquest casos de la següent manera, si dos centres eren molt propers considerarem que forma part del mateix cercle.

Un com vist el concepte que va darrera del codi passarem a parlar de com s'ha arribat a dur a terme. La funció que cridarem perquè faci tot aquest rastreig s'anomena **GetPuzzleCircles().**

```
def GetPuzzleCircles(self):
        for x in range(len(self.PixelMap)):
            if ((0,0,0) in self.PixelMap[x]):
                for y in range(len(self.PixelMap[x])):
                    #Check if it's black
                    if (self.WitchColorPixel(self.PixelMap[x][y]) == 1):
                        #Check if pixel already in a circle existing
                        if (self.NotIn([x,y])):
                            self.SearchCircleInfo([x,y],[])
                            if(len(self.Centers) == 5):
                                return 0

        return 0
```

Primer de tot el que es fa aquí és rastrejar fila per fila (eixos de les x) si es troba algun punt negre. Que com s'ha explicat anteriorment es representa (0,0,0). Sino es troba cap es torna a fer el mateix pas per la següent fila.

Si hi ha algun punt negre es busca quina és la seva o les seves posicions. Un cop trobat mirem si aquest esta en l'interior d'uns dels cercles ja trobats amb la funció **NotIn()**.

```
def NotIn(self, Point):
        for Circle in self.CirclesInfo:
            thx = (Circle[1][0]-Circle[0][0])*0.25
            thy = (Circle[2][1]-Circle[3][1])*0.25
            if ((Circle[0][0] - thx  <
Point[0])&(Circle[1][0] + thx > Point[0])&(Circle[3][1] -
thy < Point[1])&(Circle[2][1] + thy > Point[1])):
                return False
        return True
```

Si es determina que el nostre punt no està dins de cap cercle prèviament reconegut es passa a obtenir la informació del possible nou cercle amb la funció **SearchCircleInfo([x,y],[])**. Els dos paràmetres que necessita la funció són el punt des d'on es buscaran els seus marges, i l'altre és una llista amb els extrems. Al

principi serà una matriu buida, però anirà variant el seu contingut gràcies a la recursivitat.

El procediment que segueix aquest funció és el que es troba explicat amb l'explicació de la figura 15.

```
def SearchCircleInfo(self, Point, Info):
        XTop = self.FindXTop(Point)
        XDown = self.FindXDown(Point)
        YRight = self.FindYRight(Point)
        YLeft= self.FindYLeft(Point)
        #Update Circle Info
        if (Info == []):
            Info = [XTop,XDown,YRight,YLeft]
        else:
            if Info[0][0] > XTop[0]:
                Info[0] = XTop
            if Info[1][0] < XDown[0]:
                Info[1] = XDown
            if Info[2][1] < YRight[1]:
                Info[2] = YRight
            if Info[3][1] > YLeft[1]:
                Info[3] = YLeft
        center = [self.GetMiddlePoint(XTop,XDown)[0],
self.GetMiddlePoint(YRight,YLeft)[1]]
        if(self.IsValidCenter(center, Point)):
            self.CheckCenter(center, Info)
        else:
            self.SearchCircleInfo(center, Info)
```

Es comprova si aquest punt coincideix simultàniament el punt amb el mig de la línia formada per l'extrem dret i l'extrem esquerra i la que formen els extrem superior i l'inferior. Amb la funció **`IsValidCenter(center, Point)`**.

```
def IsValidCenter(self, center, point):
        XDifference = abs(point[0] - center[0])
        YDifference = abs(point[1] - center[1])
        if (((XDifference+YDifference)/2) == 0):
            return True
        return False
```

I per acabar, es comprova si el centre trobat és molt proper a un altre prèviament guardat, tal i com s'ha comentat en l'explicació de la figura 15.

Aquesta funció el que fa és calcular la diferència en les **x** i en les **y** entre el possible centre i els centre guardats, si aquesta es més petita que un marge es considera que són del mateix centre i per tant s'actualitzen les dades dels extrems si fan falta.

```
def CheckCenter(self, center, Info):
        margin = abs(round(((Info[1][0]-Info[0][0]) +(Info[2][1]-
Info[3][1]))/4.0))
        if(margin == 0):
            margin = 1
        for i in range(len(self.Centers)):
            XDifference = abs(self.Centers[i][0] - center[0])
            YDifference = abs(self.Centers[i][1] - center[1])
            if (((XDifference+YDifference)/2) < margin):
                self.Centers[i] = self.GetMiddlePoint(self.Centers[i], center)
                if self.CirclesInfo[i][0][0] > Info[0][0]:
                    self.CirclesInfo[i][0] = Info[0]
                if self.CirclesInfo[i][1][0] < Info[1][0]:
                    self.CirclesInfo[i][1] = Info[1]
                if self.CirclesInfo[i][2][1] < Info[2][1]:
                    self.CirclesInfo[i][2] = Info[2]
                if self.CirclesInfo[i][3][1] > Info[3][1]:
                    self.CirclesInfo[i][3] = Info[3]
                return 0
        self.Centers.append(center)
        self.CirclesInfo.append(Info)
```

Tot aquest procediment s'aniria repetint fins a trobar 5 centres diferents. Ara el que farà falta serà ordenar-los (P1,P2,P3,P4) per a poder utilitzar correctament a l'algoritme d'obtenció de les distàncies.

Es seguiran els següents passos:

- Buscar quins tres punts formen una línia
- D'aquest tres punts mirar quin és el del centre
- Determinar quins son el punt 2 i 3
- Determinar quins són els 4 i 1

Per determinar quins tres punts formen una línia es va decidir buscar-ho utilitzant les diferents àrees dels triangles. De tot el conjunt de triangles possibles, el que tingui l'àrea més petita serà el qui els seus tres punts formin la línia. Hi han 10 triangles possibles diferents, tal i com podem veure en la definició de la funció **`GetLinedPoints()`**.

```
def GetLinedPoints(self):
        i=0
        AreaMin = 0
        Combination = []
        while(i<10):
            if(i==0):
                AreaMin = self.CheckLine(1,2,3)
                Combination = [1,2,3]
            elif(i==1):
                a = self.CheckLine(1,2,4)
                if(a  < AreaMin):
                    AreaMin = a
                    Combination = [1,2,4]
            elif(i==2):
                a = self.CheckLine(1,2,5)
                if(a  < AreaMin):
                    AreaMin = a
                    Combination = [1,2,5]
            elif(i==3):
                a = self.CheckLine(1,3,4)
                if(a  < AreaMin):
                    AreaMin = a
                    Combination = [1,3,4]
            elif(i==4):
                a = self.CheckLine(1,3,5)
                if(a  < AreaMin):
                    AreaMin = a
                    Combination = [1,3,5]
            elif(i==5):
                a = self.CheckLine(1,4,5)
                if(a  < AreaMin):
                    AreaMin = a
                    Combination = [1,4,5]
            elif(i==6):
                a = self.CheckLine(2,3,4)
                if(a  < AreaMin):
                    AreaMin = a
                    Combination = [2,3,4]
            elif(i==7):
                a = self.CheckLine(2,3,5)
                if(a  < AreaMin):
                    AreaMin = a
                    Combination = [2,3,5]
            elif(i==8):
                a = self.CheckLine(2,4,5)
                if(a  < AreaMin):
                    AreaMin = a
                    Combination = [2,4,5]
            else:
                a = self.CheckLine(3,4,5)
                if(a  < AreaMin):
                    AreaMin = a
                    Combination = [3,4,5]
            i=i+1
        return Combination
```

Un cop trobat aquest tres punts, s'haurà de determinar quin és el punt central. D'aquest tres, s'escullen dos punts els quals creiem que son els extrem. Si el punt escollit es troba entre les dels dos punts dels extrem confirmarem que aquell és el punt del mig, sino, tornarem a fer aquest procediment fins a trobar-ne un. Aquí és on s'utilitzarà la llibreria Shapely, que ens ajudarà a determinar si un punt es troba dins o fora d'un polígon.

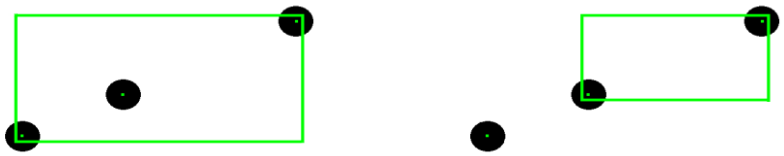


Figura 17: Detecció del punt central

Com podem veure en la figura 17 la imatge més a la dreta no compliria les condicions pel contrari la que està més a l'esquerra sí. Aquesta detecció la fem amb gràcies a la funció **`InTheMiddel(self,Im,I1,I2)`** on **`Im`** és el índex del punt del mig i **`I1`** i **`I2`** són els índex dels punts dels extrems

```
def InTheMiddel(self,Im,I1,I2):
        Pm = self.Centers[Im-1]
        P1 = self.Centers[I1-1]
        P2 = self.Centers[I2-1]
        margin = 2
        point = Point(Pm[0],Pm[1])
        margin = 2
        if(P1[0]>P2[0]):
            if(P1[1]<P2[1]):
                polygon = Polygon([(P1[0]+margin, P1[1]), (P1[0], P1[1]-
margin),(P2[0]-margin, P2[1]), (P2[0], P2[1]+margin)])
                return polygon.contains(point)
            else:
                polygon = Polygon([(P1[0]+margin, P1[1]), (P1[0],
P1[1]+margin),(P2[0]-margin, P2[1]), (P2[0], P2[1]-margin)])
                return polygon.contains(point)
        else:
            if(P1[1]<P2[1]):
                polygon = Polygon([(P1[0]-margin, P1[1]), (P1[0], P1[1]-
margin),(P2[0]+margin, P2[1]), (P2[0], P2[1]+margin)])
                return polygon.contains(point)
            else:
                polygon = Polygon([(P1[0]-margin, P1[1]), (P1[0],
P1[1]+margin),(P2[0]+margin, P2[1]), (P2[0], P2[1]-margin)])
                return polygon.contains(point)
```

Per determinar quin és el punt numero dos serà tant fàcil com calcular la distància entre els extrem i el punt del mig, la més curta serà corresponent a la del punt 2. Per aquesta tasca utilitzarem la funció **GetOrderLinedPoints(linedpoints)**, on **linedpoints** son els índex dels centres que formen la línia.

```
def GetOrderLinedPoints(self,linedpoints):
        if(self.InTheMiddel(linedpoints[1],linedpoints[0],linedpoints[2])):
            linedpoints = [linedpoints[0], linedpoints[1], linedpoints[2]]
        elif(self.InTheMiddel(linedpoints[0],linedpoints[1],linedpoints[2])):
            linedpoints = [linedpoints[1], linedpoints[0], linedpoints[2]]
        else:
            linedpoints = [linedpoints[1], linedpoints[2], linedpoints[0]]
        if(abs(self.Distance(self.Centers[linedpoints[0]-
1],self.Centers[linedpoints[1]-1])) <
abs(self.Distance(self.Centers[linedpoints[2]-1],self.Centers[linedpoints[1]-
1]))):
            return [linedpoints[2],linedpoints[0]]
        else:
             return [linedpoints[0],linedpoints[2]]
```

Ara falta trobar el punt 1 i el 4. Per això el que es fa és trobar els dos punts dels centres els quals no es troben en la línia. Amb el primer punt i el punt numero dos creem una línia, i després fem el mateix amb el segon punt i el punt numero tres. Si aquestes dues línies tenen la intersecció dins de l'objecte voldrà que el primer punt serà el punt 4 i el segon el punt 1. Sino el primer serà el punt 1 i el segon el 4. Aquesta funcionalitat està implementada en la funció **GetoOrderPoints(LinedPoints)** on **LinedPoints** son els índex dels centres que formen la línia.

```
def GetoOrderPoints(self,LinedPoints):
        PointsLeft = []
        for element in [1,2,3,4,5]:
            if element not in LinedPoints:
                PointsLeft.append(element)
        OrdenedLinedPoints = self.GetOrderLinedPoints(LinedPoints)
        intersection = self.seg_intersect(self.Centers[PointsLeft[1] -
1],self.Centers[OrdenedLinedPoints[0] - 1],self.Centers[PointsLeft[0] -
1],self.Centers[OrdenedLinedPoints[1] - 1])
        IntersectionPoint = Point(intersection[0],intersection[1])
        ObjectPoints = Polygon([(self.Centers[PointsLeft[1] - 1][0],
self.Centers[PointsLeft[1] - 1][1]),(self.Centers[PointsLeft[0] -
1][0],self.Centers[PointsLeft[0] - 1][1]), (self.Centers[OrdenedLinedPoints[0] -
1][0],self.Centers[OrdenedLinedPoints[0] - 1][1]), (self.Centers[
OrdenedLinedPoints[1]- 1][0],self.Centers[OrdenedLinedPoints[1] - 1][1])])
        if (ObjectPoints.contains(IntersectionPoint)):
            return [PointsLeft[1], OrdenedLinedPoints[1],OrdenedLinedPoints[0],
PointsLeft[0]]
        else:
            return [PointsLeft[0], OrdenedLinedPoints[1],OrdenedLinedPoints[0],
PointsLeft[1]]
```

Fent proves, testejant el treball notava que de tant en tant la distància entre dues imatges consecutives canviaven dràsticament. La càmera estava a una distància de l'objecte aproximadament de 1900 mm i quan girava una mica la càmera, de sobte deia que estava a 900 mm. Es va investigar el causant de tot això i va sortir que en uns dels centres una de les seves coordenades, ja sigui la x o la y, es modificava amb una diferència d'un píxel.

Primerament es va pensar en processar les dades dels centres abans que es tractessin per obtenir la distància. Una de les idees va ser reconstruir el nostre objecte perquè formes un paral·lelogram, si els centres no en formaven un. Amb aquest canvi vam notar una millora substancial per la part de les distàncies. Ara bé per l'orientació no tant, ja que sempre l'orientació en els eixos dels eixos en les x i les y eren estables a 180 i a 0 respectivament.

5.3. Mòdul per obtenir les distàncies

Aquest mòdul s'anomena **caclulate_coordinates.py** i és on s'han implementat les diferents equació ja explicades en el l'apartat (*4. Algoritme)*.

El mòdul conté una classe Python anomenada Enviroment3d amb diferents funcions que ens permetran calcular totes les equacions.

```
def __init__(self,s12,s13,s14,s23,s24,s34):
        self.CameraCenterPrint = [75,75,600]
        self.CameraCenter = np.array([[75],[75],[600]])
        self.HeightPixels = 384
        self.WidthPixels = 512
        self.Pixel2mmX = ( 1944 * 1.4*10**(-3))/self.HeightPixels
        self.Pixel2mmY = ( 2592 * 1.4*10**(-3))/self.WidthPixels
        self.RotationMatrix = []
        self.Focal = 3.6
        self.s12 = s12
        self.s13 = s13
        self.s14 = s14
        self.s23 = s23
        self.s24 = s24
        self.s34 = s34
        self.points = []
        self.d1
        self.d2
        self.d3
        self.d4
```

- **CameraCenterPrint, CameraCenter:** Punt on es troba la càmera, aquestes variables les hem utilitzats només per a debugar

- **RotationMatrix:** Matriu de rotació de la càmera, igual que la anterior, aquesta variable està creada per a debugar.
- **HeightPixels:** Nombre de píxels que té d'alçada la foto on es troba l'objecte.
- **WidthPixels :** Nombre de píxels que té d'amplada la foto on es troba l'objecte.
- **Pixel2mmX:** Variable que determina el valor en mil·límetres que equival cada píxel en els eixos de les X.
- **Pixel2mmY:** Variable que determina el valor en mil·límetres que equival cada píxel en els eixos de les y.
- **Focal:** Distància focal de la càmera, si en té.
- **s12, s13, s14, s23 , s24, s34:** Els 6 paràmetres que determinen les mesures del nostre objecte. Aquest s'hauran de passar quan es crea aquest objecte.
- **Points:** Llista dels 4 punts expressats en funció de l'eix de coordenades de la càmera.
- **d1, d2, d3, d4:** Les 4 diferents distàncies que hi ha del centre de la càmera al punt.

Primer de tot el que farem serà passar els punts, que vindran donats en píxels, a mil·límetres i després s'afegiran a la variable **Points**.

```
def Pixel2Camera(self, Points):
        result = []
        for point in Points:
             result.append([(( self.HeightPixels/2) -
point[0]) * self.Pixel2mmX,(point[1] –
(self.WidthPixels/2) )*self.Pixel2mmY])
        return result

def AddPointsTest(self,result):
        for point in result:
            self.AddPointsPicture(point[0],point[1])
```

Per facilitar la visualment l'escriptura de les equacions s'ha decidit definir les diferents coordenades dels punts de la següent manera.

x1() Aquesta funció retorna la coordenada x del $\boldsymbol{P_1}$

y1() Aquesta funció retorna la coordenada y del $\boldsymbol{P_1}$

x2() Aquesta funció retorna la coordenada x del $\boldsymbol{P_2}$

y2() Aquesta funció retorna la coordenada y del $\boldsymbol{P_2}$

x3() Aquesta funció retorna la coordenada x del $\boldsymbol{P_3}$

y3() Aquesta funció retorna la coordenada y del $\boldsymbol{P_3}$

x4() Aquesta funció retorna la coordenada x del $\boldsymbol{P_4}$

y4() Aquesta funció retorna la coordenada y del $\boldsymbol{P_4}$

Per calcular les diferents àrees $\boldsymbol{A_i}$ que es troben en les equacions(2.8,...,2.11) es farà gràcies a aquestes funcions.

A1() Aquesta funció retorna l'àrea del triangle format per $\boldsymbol{P_1P_2P_3}$

A2() Aquesta funció retorna l'àrea del triangle format per $\boldsymbol{P_1P_2P_4}$

A3() Aquesta funció retorna l'àrea del triangle format per $\boldsymbol{P_1P_4P_3}$

A4() Aquesta funció retorna l'àrea del triangle format per $\boldsymbol{P_4P_2P_3}$

Per calcular les diferents $\boldsymbol{B_i}$ que es troben en les equacions(2.25,...,2.28) es farà gràcies a aquestes funcions.

B1() Aquesta funció retorna dos vegades l'àrea del triangle format per$\boldsymbol{P_1P_2P_3}$

B2() Aquesta funció retorna dos vegades l'àrea del triangle format per $\boldsymbol{P_1P_2P_4}$

B3() Aquesta funció retorna dos vegades l'àrea del triangle format per $\boldsymbol{P_1P_4P_3}$

B4() Aquesta funció retorna dos vegades l'àrea del triangle format per $\boldsymbol{P_4P_2P_3}$

Per calcular les diferents $\boldsymbol{C_{ij}}$ que es troben en les equacions(2.29,...,2.34) es farà gràcies a aquestes funcions.

C12() Aquesta funció retorna $\boldsymbol{C_{12}}$

C13() Aquesta funció retorna $\boldsymbol{C_{13}}$

C14() Aquesta funció retorna $\boldsymbol{C_{14}}$

C23() Aquesta funció retorna $\boldsymbol{C_{23}}$

C24() Aquesta funció retorna $\boldsymbol{C_{24}}$

C34() Aquesta funció retorna $\boldsymbol{C_{34}}$

Per calcular les diferents $\boldsymbol{H_{ij}^2}$ que es troben en l'equació(2.40) es farà gràcies a aquestes funcions.

H12_2() Aquesta funció retorna $\boldsymbol{H_{12}^2}$

H13_2() Aquesta funció retorna $\boldsymbol{H_{13}^2}$

H14_2() Aquesta funció retorna $\boldsymbol{H_{14}^2}$

H23_2() Aquesta funció retorna $\boldsymbol{H_{23}^2}$

H24_2() Aquesta funció retorna $\boldsymbol{H_{24}^2}$

H34_2() Aquesta funció retorna $\boldsymbol{H_{34}^2}$

Per calcular les diferents $\boldsymbol{F_i}$ que es troben en l'equació(2.18) es farà gràcies a aquesta funció.

Fi(point) Aquesta funció retorna la magnitud del vector $\boldsymbol{Q_i^f}$, on **point** es el numero del punt. **point** pot tenir els següents valors{1,2,3,4}

Per calcular les diferents $\boldsymbol{d_i}$ que es troben en les equacions(2.41, 2.29, 2.30, 2.31) es farà gràcies a aquestes funcions.

d1() Aquesta funció retorna la distancia $\boldsymbol{d_1}$

d2() Aquesta funció retorna la distancia $\boldsymbol{d_2}$

d3() Aquesta funció retorna la distancia $\boldsymbol{d_3}$

d4() Aquesta funció retorna la distancia $\boldsymbol{d_4}$

Per calcular les diferents $\boldsymbol{PC_i}$ que es troben en les equacions(2.41,...,2.46) es farà gràcies a aquestes funcions.

PC1() Aquesta funció retorna les coordenades del punt $\boldsymbol{PC_1}$

PC2() Aquesta funció retorna les coordenades del punt $\boldsymbol{PC_2}$

PC3() Aquesta funció retorna les coordenades del punt $\boldsymbol{PC_3}$

PC4() Aquesta funció retorna les coordenades del punt $\boldsymbol{PC_4}$

Per acabar, per calcular , la matriu T que es troben definida en les equacions(2.51,..., 2.62) es farà gràcies a aquesta funció.

Tmatrix1(P1,P2,P3,P4) Aquesta funció omplirà la variable **self.T** amb la translation matrix corresponent. **P1,P2,P3,P4** són els diferents $\boldsymbol{P_i}$. En el nostre cas tindran els següents valors sempre.

```
P1 = [0,0,0], P2 = [40,0,0]
P3 = [40,100,0] , P4 = [0,100,0]
```

Per a obtenir definitivament el $\boldsymbol{X_0, Y_0, Z_0, \alpha, \beta, \gamma}$ el que es farà serà cridar aquesta funció:

GetPositonXYZ() Aquesta funció gràcies a la matriu **self.T** retornarà una llista amb el valors[$\boldsymbol{X_0, Y_0, Z_0, \alpha, \beta, \gamma}$].

Dins d'aquesta funció anterior s'utilitza una funció extreta d'un mòdul implementat pel Christoph Gohlke que podreu trobar en [9]. Aquest mòdul ofereix una gran varietat de funcions Python per a treballar amb matrius de rotacions, translacions, escalats i per suposat la tranformation matrix. La que utilitzarem nosaltres s'anomena:

> **`decompose_matrix(matrix)`** Aquesta retorna la seqüència de transformacions he ha patit la matriu **`matrix`**. D'aquesta seqüència només volem saber el contingut de les de translació i la del angles, aquest angles estan expressats amb radiants.

Fent moltes proves, es va veure que la translació($\boldsymbol{X_0}, \boldsymbol{Y_0}, \boldsymbol{Z_0}$) no era la que esperàvem. Aquestes coordenades expressaven l'origen de coordenades de l'objecte en funció dels eixos de coordenades de la càmera. El que volíem era el contrari, el origen de la càmera en funció dels eixos de coordenades de l'objecte.

Això es soluciona fàcil entenent bé el funcionament de la matriu T.

$$W^C = TW$$

$$O_c = TC$$

On $\boldsymbol{O_c}$ és l'origen de coordenades de la càmera en funció els eixos de coordenades de la càmera, per tant [0,0,0], i $\boldsymbol{C}$ respecte l'objecte

$$O_c\, T^{-1} = C$$

$$O_c\, 'T^{-1\prime} = C'$$

$$[0\ \ 0\ \ 0\ \ 1]T^{-1\prime} = \ [X_1^O\ \ Y_1^O\ \ Z_1^O\ \ 1]$$

5.4. Estructura del directori de treball

Dins de [13] trobarem el directori git que s'ha utilitzat per estructurar els treball. La carpeta arrel conte el fitxer **`Global.Main.py`** el qual es el fitxer que executarem perquè el sistema engegui i els fitxers de test que s'han anat fent per provar els diferents mòduls i el funcionament del propi programa. A part d'això hi trobarem 2 directoris més:

- **`Img/`**: Aquest serà on es guardaran les diferents fotos.
- **`src/`**: Aquí hi trobarem els mòduls per que els sistema funcioni

Dins d'aquest del directori **`src`** trobarem els següents mòduls:

- frameV2.py
- calculate_coordinates.py
- img.py
- transformation.py

5.5. Proves i testos

5.5.1. Reconeixement de l'objecte

Per a testejar aquesta part s'utilitza el fitxer ***`mainV2.py`***. Aquest, com a argument necessita el path de la imatge la qual vulguis trobar el centres. El que et permet visualitzar pel terminal es:

- Temps de processat de la imatge en segons
- Les coordenades del 5 centres
- L'índex dels punts ordenats (P1, P2, P3, P4)
- Coordenades dels punts ordenats

A part d'això també es creen dues imatges en el directori ***`./Img`***. Una és per poder veure el resultat de la imatge filtrada pel valor del ***`threshold`*** i l'altre per visualitzar el centres i els extrems de cada cercle. Seguidament es poden veure uns quants exemples del resultats obtingut de diverses figures 18,19 i 20:

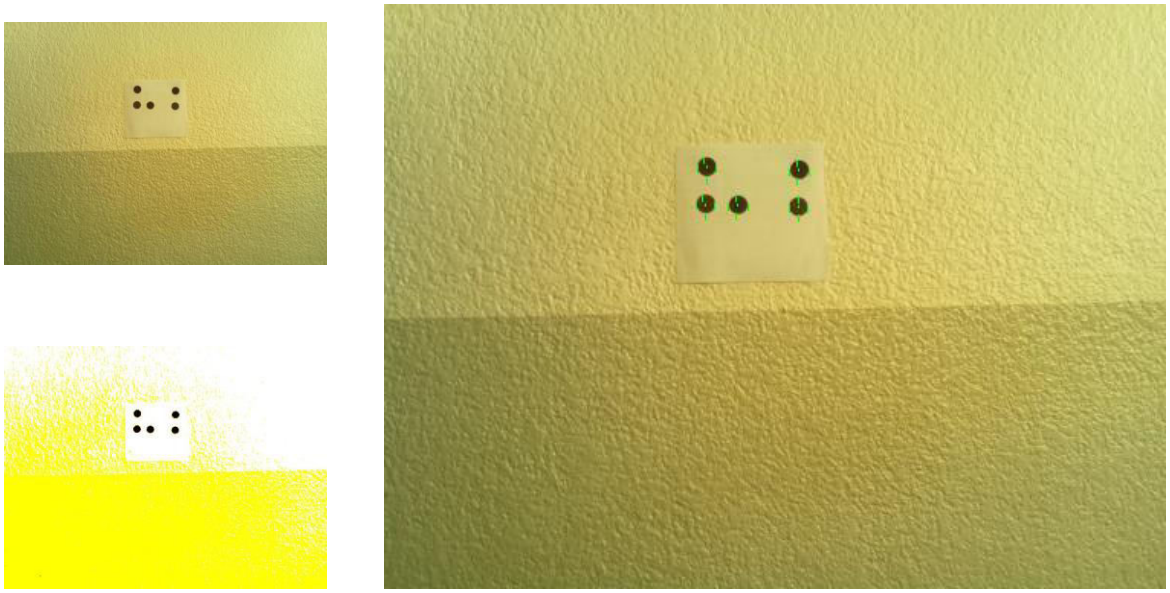


Figura 18: Resultats obtinguts de la primera prova

```
--- 0.00469398498535 seconds ---
Hi han [[106, 211], [108, 271], [131, 210], [131, 231], [132, 271]]
Index centre ordenats: [1, 3, 5, 2]
Centres ordenats: [[106, 211], [131, 210], [132, 271], [108, 271]]
```

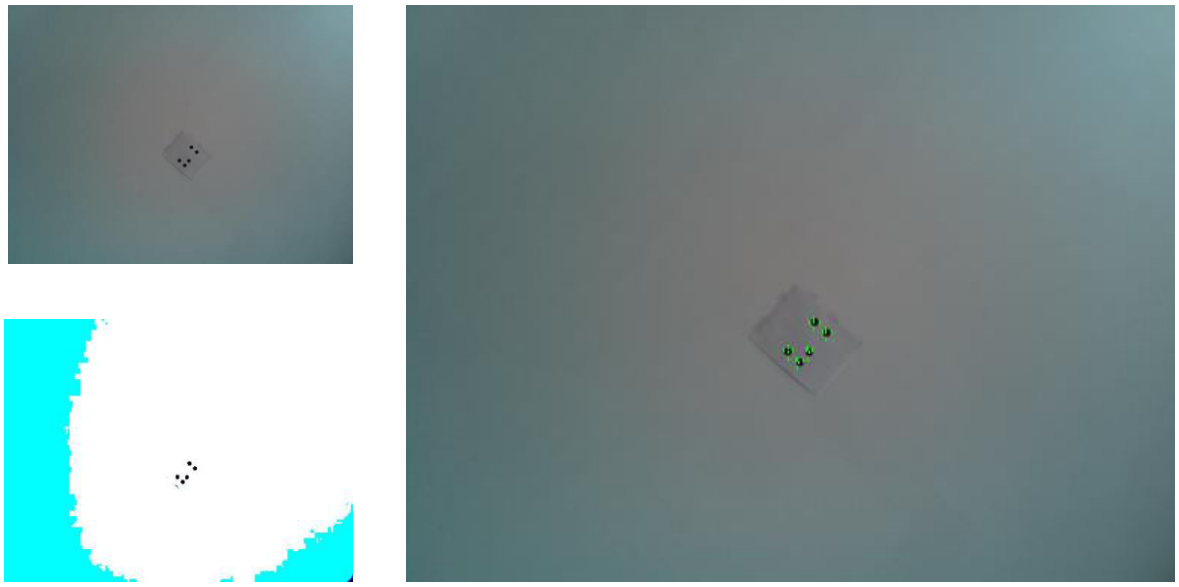


Figura 19: Resultats obtinguts de la segona prova

```
--- 0.00626587867737 seconds ---
Hi han [[210, 271], [217, 279], [230, 254], [230, 268], [237, 261]]
Index centre ordenats: [3, 5, 2, 1]
Centres ordenats: [[230, 254], [237, 261], [217, 279], [210, 271]]
```

Figura 20: Resultats obtinguts de la tercera prova

```
--- 0.012638092041 seconds ---
Hi han [[300, 151], [304, 177], [348, 169], [367, 139], [371, 165]]
Index centre ordenats: [4, 5, 2, 1]
Centres ordenats: [[367, 139], [371, 165], [304, 177], [300, 151]]
```

5.5.2. Obtenció de l'orientació

Per a testejar aquesta part hem utilitzat el fitxer ***test_transform.py***. En aquest hi han tres variables que s'hauran d'omplir en funció de la posició i orientació de la càmera:

- $\boldsymbol{c}$: llista on hi trobarem la translació que ha patit la càmera [X, Y, Z]
- $\boldsymbol{rot}$: llista on hi trobarem la rotació que ha patit la càmera [$\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$]
- $\boldsymbol{f}$: distància focal de la càmera

La idea que hi ha al darrera aquest fitxer de proves es simular virtualment on es trobarien els punts en el pla de la càmera , per després tractar-los i obtenir els resultats inicials. A part d'això també el farem per veure quines diferències hi hauran si un dels centres està lleugerament desplaçat. El resultats que obtindrem a través del terminal seran:

- Les coordenades dels punts en pla de la càmera, i els punts si s'han desplaçat
- Les $\boldsymbol{X_0}, \boldsymbol{Y_0}, \boldsymbol{Z_0}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ trobades.

A continuació podem veure un exemple de l'aspecte d'una resposta. Totes les unitats dels valors que es mostraran a continuació, tret dels angles referenciats amb (º), seran mil·límetres.

```
[-0.8652699617979418, -0.24097630784027618] to [-
0.85818246179794178, -0.24097630784027618]
[-0.77070502273015318, 0.111944849375366] to [-0.77070502273015318,
0.111944849375366]
[0.10589859911816599, -0.12089441799222485] to [0.10589859911816599,
-0.12089441799222485]
[0.013379625239240918, -0.46617992917251105] to
[0.013379625239240918, -0.46617992917251105]
**************************
X0 = 50.0
Y0 = 50.0
Z0 = 400.0
a = 175.0
b = -0.0
c = 75.0
**************************
```

En la taula 3 veurem els diferents resultats que s'obtenen en funció de la rotació i la translació de la càmera entrats inicialment sense cap centre desplaçat. Com podem observar els resultats obtinguts són tots correctes.

PARÀMETRES OROGINALS						PARAMETRES RESULTANTS					
X	Y	Z	a	b	c	X	Y	Z	a	b	c
50	50	400	175º	0º	75º	**50**	**50**	**400**	**175º**	**0º**	**75º**
320	116	600	180º	10º	75º	**320**	**116**	**600**	**180º**	**10º**	**75º**
320	116	2600	160º	-25º	90º	**320**	**116**	**2600**	**160º**	**-25º**	**90º**

Taula 3

En la taula numero 4 veurem els resultats obtinguts si un dels centres es troba desplaçat. Els resultats quan ens trobem a una distància curta, uns 200 mm, l'error que generat no és gaire gran. Però quan augmentem la distància els valor ja no són correctes.

PARÀMETRES ORIGINALS						PARAMETRES RESULTANTS					
X	Y	Z	A	b	c	X	Y	Z	a	b	c
50	50	200	180º	0º	0º	**49**	**45.8**	**199.4**	**178.9º**	**-0.19º**	**9.95º**
50	50	500	180º	0º	0º	**48.6**	**-10.5**	**491.5**	**173.1º**	**0º**	**0º**
50	50	1000	180º	0º	0º	**39.6**	**-341.3**	**870.8**	**154.3º**	**0º**	**0º**

Taula 4

Els resultats obtinguts de la taula 4, s'han trobat modificant les x pel punt número dos. Els resultats varien en funció de quin punt es modifica i en quina orientació es troba la càmera. Després de moltes proves fetes he determinat que quan la rotació en l'eix X es propera a 180^o l'error tendeix a augmentar.

Passem a veure com afecta aquest desplaçament dels centre amb les distàncies entre els diferents quatre punts i la càmera. Els resultats els trobarem a la taula 5. Aquesta vegada els errors no superen el 10% en totes menys en l'últim. Per tant el rang del nostre sistema com a màxim arribarà sobre els 900 mil·límetres.

PARÀMETRES ORIGINALS						PARAMETRES RESULTANTS Sense desplaçament				PARAMETRES RESULTANTS Amb desplaçament			
X	Y	Z	a	b	c	D1	D2	D3	D4	D1	D2	D3	D4
50	50	200	180^o	0^o	0^o	212.1	206.4	206.4	212.1	**210.8**	**205.1**	**207.1**	**212.8**
50	50	500	180^o	0^o	0^o	504.9	502.6	502.6	504.9	**494**	**491.6**	**503.7**	**506.2**
50	50	700	180^o	0^o	0^o	703.6	701.9	701.9	703.6	**675.3**	**673.7**	**696.9**	**698.6**
50	50	900	180^o	0^o	0^o	902.8	901.4	901.4	902.8	**828.2**	**826.9**	**863.6**	**864.8**
50	50	1000	180^o	0^o	0^o	1002.5	1001.3	1001.3	1002.5	**890.9**	**889.9**	**933.7**	**934.8**

Taula 5

Detectarem si una mesura és correcte, per tant si la translació i rotació són correctes, gràcies a uns factors. Aquest sis factors estan descrits en l'equació (*2.42*). Aquest varien en funció de les deformacions del pla de la càmera. Considerarem que mesura correcte quan els sis factors tinguin el mateix valor.

6. Hardware

6.1 Ordinadors monoplaca

Es va decidir utilitzar la Raspberry Pi 3 Model B[10], ja que es un dispositiu el petit, lleuger i ens permetrà processar el nostre programa. A part d'això permet una integració molt fàcil alhora d'utilitzar una càmera amb llibreries ja preparades i a més a més es un dels dispositius qui s'utilitzen bastant dintre de l'àmbit de l'enginyeria en sistemes TIC. Aquesta té principalment les següents característiques:

- Processador quad core 1.2GHZ Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN i Bluetooth Low Energy (BLE)
- CSI camera port for connecting a Raspberry Pi camera

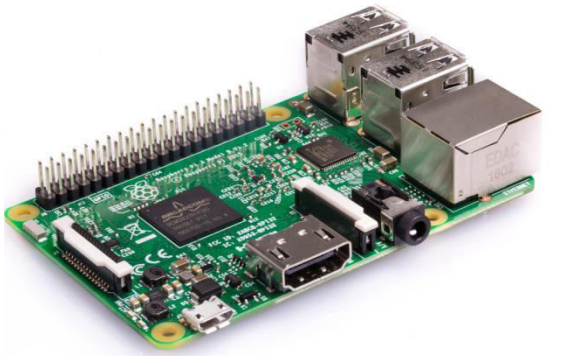


Figura 18: Raspberry PI 3 Model B

6.2 Càmera

Com s'ha comentat prèviament, la Raspberry que farem servir, permet utilitzar un model de càmera que es pot integrar molt fàcilment. Aquest model se'n diu RaspiCam[11]. Aquesta anirà connectada al port CSI camera i té principalment aquestes característiques:

- 5 Megapixels
- Resolució màxima de 2592 x 1944 píxels
- Mida dels píxels 1.4 µm × 1.4 µm
- Distància focal de 3.60 mm +/- 0.01

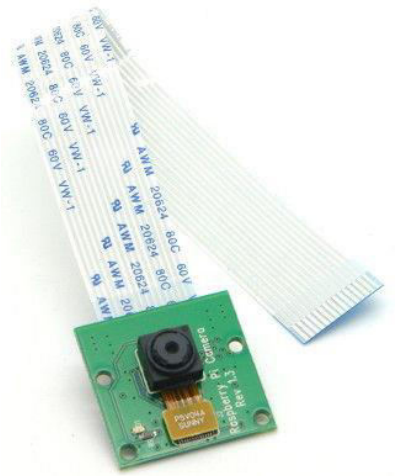


Figura 19: RaspiCam

6.3 Muntatge final

Tal i com podem veure a la figura 19 el cablejat blanc és molt flexible per tant era necessària una estructura. El primer muntatge que es va realitzar va ser un amb una estructura feta amb peces de Lego, on s'hi subjectava la càmera i la Raspberry. Aquest va ser molt útil durant les primeres proves per mesurar les distàncies que hi havia entre els punts i la càmera. El primer muntatge el podem veure en la figura 20.

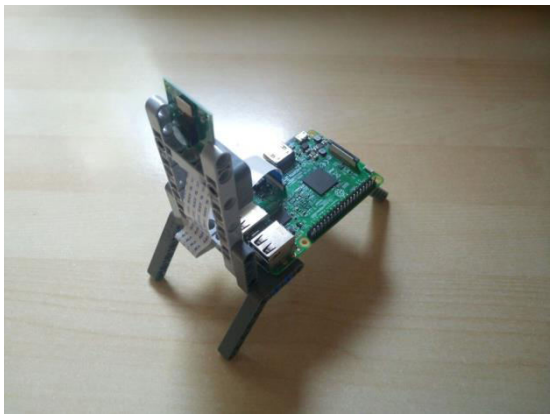


Figura 20: Primera estructura

Però per provar l'orientació de la càmera, per tant els girs aplicats, aquesta no ens anava massa be ja que la part on estava posicionada la càmera era molt fixe. Per tant es va decidir utilitzar un trípode per fer més àgils els girs. Però necessitàvem una nova estructura per poder fixar–la al trípode. Per tant vam decidir imprimir-nos un objecte en 3D[12]. El muntatge final el podem veure a la figura 21.

Figura 21: Muntatge final

7. Integració i testos

El sistema operatiu que utilitzarà la Raspberry Pi és el Raspbian, una distribució de Linux feta especialment pels ordinadors monoplaca de Raspberry Pi. Les imatges d'aquest sistema operatiu la podeu trobar a [15].

Perquè funcioni el nostre projecte haurem d'habilitar la càmera en la Raspberry PI [16] i baixar-se el directori de treball [13].

Podrem accedir de dues maneres al arxius de la Raspberry Pi, una és connectant un monitor pel port HDMI i un teclat pel port usb, o bé remotament. En el nostre cas hem utilitzat la connexió remota per dues vies diferents:

- SSH [17]: Aquest tipus de connexió es farà via el terminal. Aquesta en permetrar accedir i manejar la Raspberry Pi. Per tant podem executar el nostre programa sense problema. El que no podrem fer serà visualitzar continguts gràfica com ara les fotos.
- SFTP [18]: Aquest tipus de connexió es farà via el gestor d'arxius en l'apartat on diu "Connecta a un servidor". Aquesta ens permetrar tant visualitzar com modificar i copiar els arxius de dins de la Raspberry Pi.

Tan per la connexió via SSH com amb la SFTP és necessari saber la adreça Ip, un usuari i la seva contrasenya. Normalment el nom d'usuari i contrasenya que es troba per defecte en el Raspbian OS és:

- Usuari : pi
- Contrasenya: raspberry

Per dur a terme la connexió per SSH el que farem serà obrir el terminal i escriure:

ssh usuari@adreçaIp

On diu **usuari** hi posarem en nom d'usuari i **adreçaIp** l'adreça Ip. Després ens demanarà la contrasenya i si és correcta ja tindríem la connexió establerta.

L'explicació per dur a terme una connexió SFTP la faré en base el sistema operatiu Ubuntu 14.04 LTS. Obrirem el gestor d'arxius i clicarem on posa "Connectar a un servidor" a l'apartat de xarxa, com podem veure en la figura 22. Després on posa Adreça del servidor escrivim:

sftp://adreçaIp/home/usuari

Com hem comentat abans, on diu **usuari** hi posarem en nom d'usuari i **adreçaIp** l'adreça Ip. Un com ho tinguem escrit clicarem el botó "Connectar". Seguidament ens

demanarà l'usuari i contrasenya, introduirem els camps i si són correctes ja podrem veure el contingut del directori arrel del usuari.

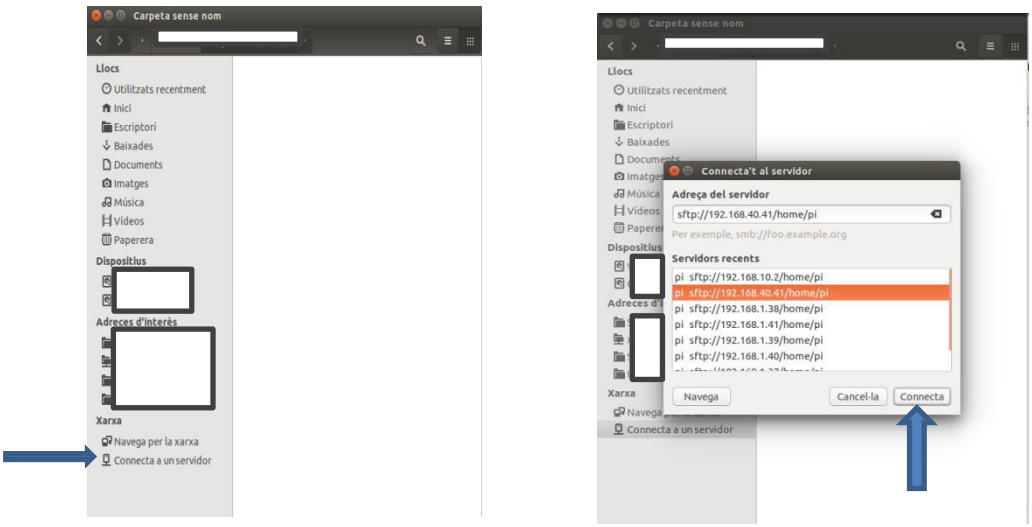


Figura 22: Procediment per connexió SFTP

És molt important connectar la càmera a la Raspberry Pi abans que s'engegui, ja que sino no la detecta. Per tant quan el nostre programa intenti fer una foto sortirà un error.

El fitxer on s'han integrat els dos mòduls, **`frameV2.py`** i **`calculate_coordinates.py`** , és l'anomenat **`GlobalMain.py`**. El esquema que seguirà és el següent:

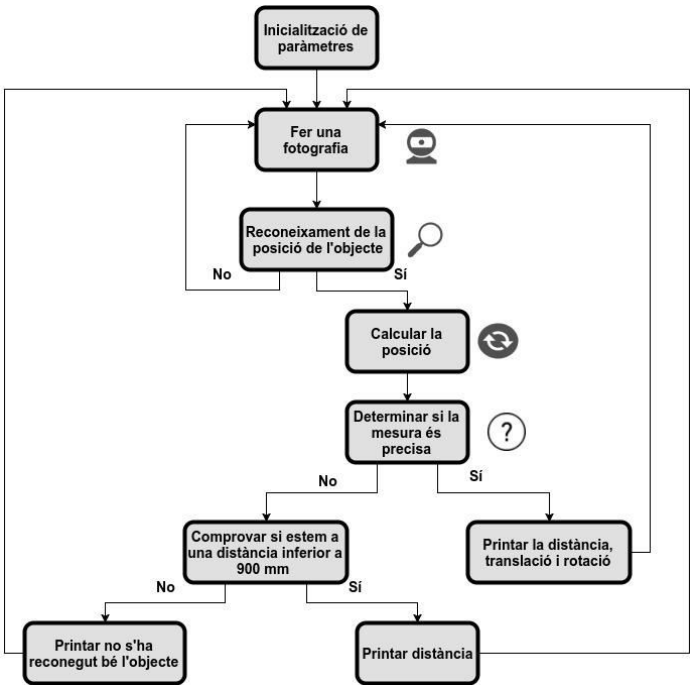


Figura 22: Esquema del funcionament del programa principal

Tota la informació que es va extraient del programa es veure representada en el mateix terminal on l'hem iniciat.

El lloc on hem fet les proves de funcionament és el que veiem en la figura 23. El nostre objecte de referència enganxat a una paret de color clar amb la Raspicam mirant-la.

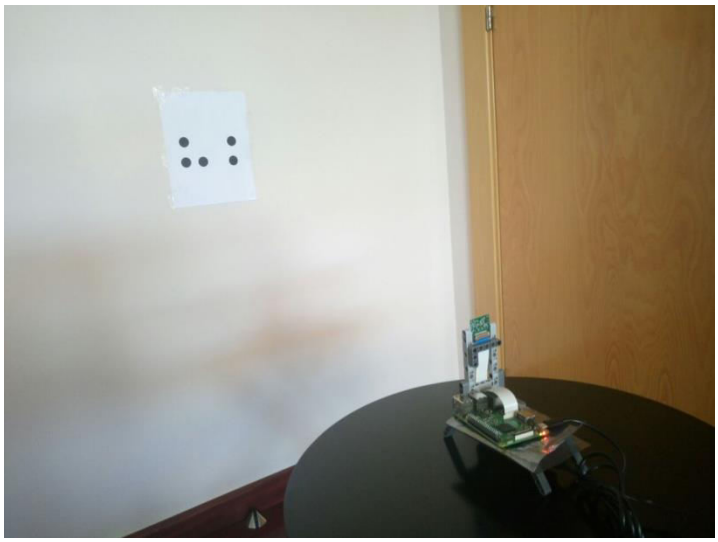


Figura 23: Muntatge per fer les proves del sistema

Com hem mencionat anteriorment les proves les farem en funció de la distància, ja que calcular la translació ens vam trobar que era bastant complicat. Per comprovar les distàncies, utilitzarem una cinta mètrica per mesurar-la.

DISTÀNCIA MESURADA DE LA CÀMERA AL PUNT P1	RESULTAT DISTÀNCIA CÀMERA AL PUNT P1	RESULTAT DISTÀNCIA CÀMERA AL PUNT P2	RESULTAT DISTÀNCIA CÀMERA AL PUNT P3	RESULTAT DISTÀNCIA CÀMERA AL PUNT P4
280	277.6	244.2	254.4	282.63
400	398.91	378.95	369.7	389.8
525	532.8	519.4	505.5	527
690	696.2	686.2	691	697.9
885	834.4	831.7	863.2	866.4

Taula 6

La taula 6 mostra les distàncies en mil·límetres les distàncies i es corresponen força a la realitat. Un dels problemes que he notat que en cap moment es mostrava la translació i rotació. Per tant, si mirem l'esquema de la figura 22, significa que en cap cas les dades extres no són precises.

Un dels problemes pel quan les dades no siguin del tot precises podria ser que el software de reconeixement del l'objecte no sigui gaire precís, ja que tal i com hem vist en l'apartat (5.2.2. *Obtenció de l'orientació*) amb una diferència d'un píxel les dades varien significativament.

Un altre, que crec que és el causant, és pel fet de la distorsió en la imatge produïda per les lents. Això es podria corregir amb un calibrat previ de la càmera. Aquest calibrat permet trobar aquestes distorsions per llavors poder rectificar-les quan volem transformar la posició del píxel en mil·límetres. En moltes llibreries el calibratge previ de la càmera era necessari, però es va pensar que si aquesta llibreria es feia des de zero no caldria.

Un altre aspecte a parlar sobre els resultats és el temps de processat de cada imatge. Cada cicle, entre fer la foto, processar-la i extreure la imatge tarda aproximadament 2 segons. Aquest és un temps massa alt per un sistema que es volia dissenyar que fos a temps real.

8. Conclusions

Per concloure el treball, m'agradaria explicar el resultat final. S'ha aconseguit un sistema on, a la pràctica, determina a quina distància es troba la càmera de l'objecte amb una distància màxima de 900 mil·límetres. Les dades s'obtenen cada 2 segons. Per altre banda tenim dissenyat un software que ens permetrà determinar la translació i la orientació de la càmera si les mesures són precises.

Aquestes característiques no compleixen massa amb les que s'havien pensat inicialment. El programa hauria de ser més ràpid analitzant i més precis en la mesura dels centres.

M'agradaria també mencionar que buscar informació sobre com poder fer un programa de posicionament amb Python sense utilitzar llibreries ha estat bastant complicat. Es va voler crear una pròpia llibreria per poder augmentar la velocitat de processament, però al final no s'ha aconseguit.

Per acabar aquestes conclusions, direm que s'ha aconseguit un sistema de posicionament en interiors que mesura les distàncies entre la càmera i els punts. Aquest no compta amb l'orientació ja que les dades extretes no son del tot precises.

Com a possibles millores de cara a projectes futurs proposaria les següents:

- Creació d'un mòdul que permetés calibrar la càmera per poder millorar la precisió de les mesures.
- Millorar el procediment de detecció de l'objecte, que poguessin haver més punts negres i determinar quin son els que formen l'objecte de referència.

Bibliografia

1. Chandra C. Tan. (1989). Landmark tracking and camera calibration. Retrieved from https://www.imaging.utk.edu/publications/papers/dissertation/tan.pdf
2. Hartley, R. I., & Sturm, P. (n.d.). Triangulation. Retrieved from http://users.cecs.anu.edu.au/~hartley/Papers/triangulation/triangulation.pdf
3. Transformation matrix. Retrieved May 24, 2017, from https://en.wikipedia.org/wiki/Transformation_matrix
4. Installation — Pillow (PIL Fork) 4.1.1 documentation. (n.d.). Retrieved June 15, 2017, from http://pillow.readthedocs.io/en/4.1.x/installation.html
5. Installation — Pillow (PIL Fork) 4.1.1 documentation. (n.d.). Retrieved June 15, 2017, from http://pillow.readthedocs.io/en/4.1.x/installation.html
6. Jones, D. (2017). Picamera 1.13 Documentation. Retrieved from https://media.readthedocs.org/pdf/picamera/latest/picamera.pdf
7. Image Module — Pillow (PIL Fork) 4.1.1 documentation. (n.d.). Retrieved June 19, 2017, from https://pillow.readthedocs.io/en/4.1.x/reference/Image.html
8. Shapely 1.6b4 : Python Package Index. (n.d.). Retrieved June 25, 2017, from https://pypi.python.org/pypi/Shapely
9. Christoph Gohlke. transformations.py. Retrieved June 28, 2017, from http://www.lfd.uci.edu/~gohlke/code/transformations.py.html
10. Raspberry Pi 3 Model B - Raspberry Pi. (n.d.). Retrieved July 1, 2017, from https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
11. Raspberry Pi 3 Model B - Raspberry Pi. (n.d.). Retrieved July 1, 2017, from https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
12. Raspberry Pi 3 Model B - Raspberry Pi. (n.d.). Retrieved July 1, 2017, from https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
13. Robert Figueroa Moreno . Retrieved July 4, 2017, from https://github.com/RobStak/IndoorPositioning
14. López Magem, C. (2015). Sistema de Posicionament en Interiors.
15. Download Raspbian for Raspberry Pi. (n.d.). Retrieved July 6, 2017, from https://www.raspberrypi.org/downloads/raspbian/
16. Camera configuration - Raspberry Pi Documentation. (n.d.). Retrieved July 6, 2017, from https://www.raspberrypi.org/documentation/configuration/camera.md
17. Secure Shell - Wikipedia, la enciclopedia libre. (n.d.). Retrieved July 6, 2017, from https://es.wikipedia.org/wiki/Secure_Shell
18. SSH File Transfer Protocol - Wikipedia, la enciclopedia libre. (n.d.). Retrieved July 6, 2017, from https://es.wikipedia.org/wiki/SSH_File_Transfer_Protocol